パソコンプログラマのためのクリエイティブマガジン

平成3年4月2日第三種郵便物認可　1992年6月1日発行(毎月1回18日)　第2巻第6号通巻18号

月刊プログラマーズ・ページ

Programmer's Page

MONTHLY MAGAZINE FOR PROGRAMMERS

1992 6 JUNE

定価980円

特集

アセンブラの作り方

8086系簡易アセンブラの設計と作成

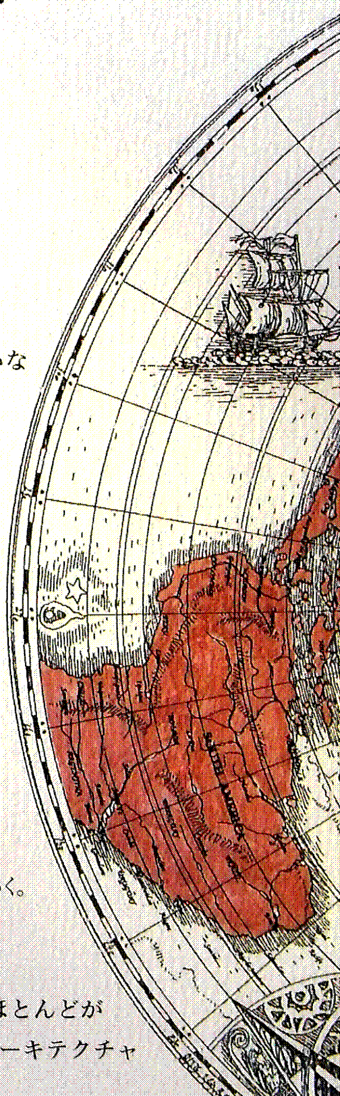
付録:5”2HDフロッピーディスク

本誌掲載プログラム（特集「8086系簡易アセンブラ」ほか）／綴じ込み付録テキストファイル（8086インストラクション一覧）／クイズ・ランダムスクリーン解読ツール／フロッピーディスクファイル化ツール「MAHALITO ver.2.10」

SE
Shoeisha Co., Ltd.

世界標準物語　第三話

14本目の煙草



とある企業のオフィス。二人の男がテーブルをはさみ、真剣な顔で向き合っている。

テーブルの上の灰皿には、すでに12本の煙草の吸殻がある。

A部長「それじゃあキミはどうしても、AXがいいというんだね？」

X主任「はい、どうしてもAXがいいんです。今回だけは一歩も譲れません」

短い沈黙。A部長、13本目の煙草を取り出し、おもむろに火をつける。

A部長「しかし、うちの社内でAXを使っているのは、まだキミを含めて数人しかいないんだ。他のものが使いこなせるかね」

X主任「使いこなせるもなにも、操作自体は全く変わらないんです。違うのは、パソコン内部のアーキテクチャです。AXは、事実上の世界標準機であるIBMPC/ATと同じオープンアーキテクチャを採用しているので、世界に6万本以上といわれるIBMPC用ソフトがそのまま使えるんです。今、うちの社内で使っているパソコンは、日本では一応、標準機ということになってますが、閉ざされた、メーカー独自のアーキテクチャを採用しているので、そういうことはないんですよ」

A部長「IBMPCは、そんなに世界中で使われているのか？」

X主任、待ってましたとばかりに、一枚の世界地図をとりだす。

X主任「見てください。アメリカはもちろん、カナダ、メキシコ、ブラジル、イギリス、フランス、ドイツ、ソ連、中国…………。IBMPCとその互換機は日本以外のほとんどの国でトップシェアを占めているのが現実です」

X主任、つぎつぎに国名をあげながら、猛スピードでその国の位置を赤鉛筆で塗りつぶしていく。

A部長、思わず大きな溜息とともに煙を吐く。

A部長「うーん、これはすごい。日本はまさに世界の異端児ってわけだ」

X主任「MS-Windows、LAN、DTPソフトなど、現在我々が使っているソフトのほとんどが海外でIBMPC用として開発されたものです。日本語への移植も、AXなら基本アーキテクチャが同じなので即座に行えるというわけです」

A部長「GUI環境を実現するWindows3.0対応ソフトがすぐ使えるのは、確かに魅力だな」

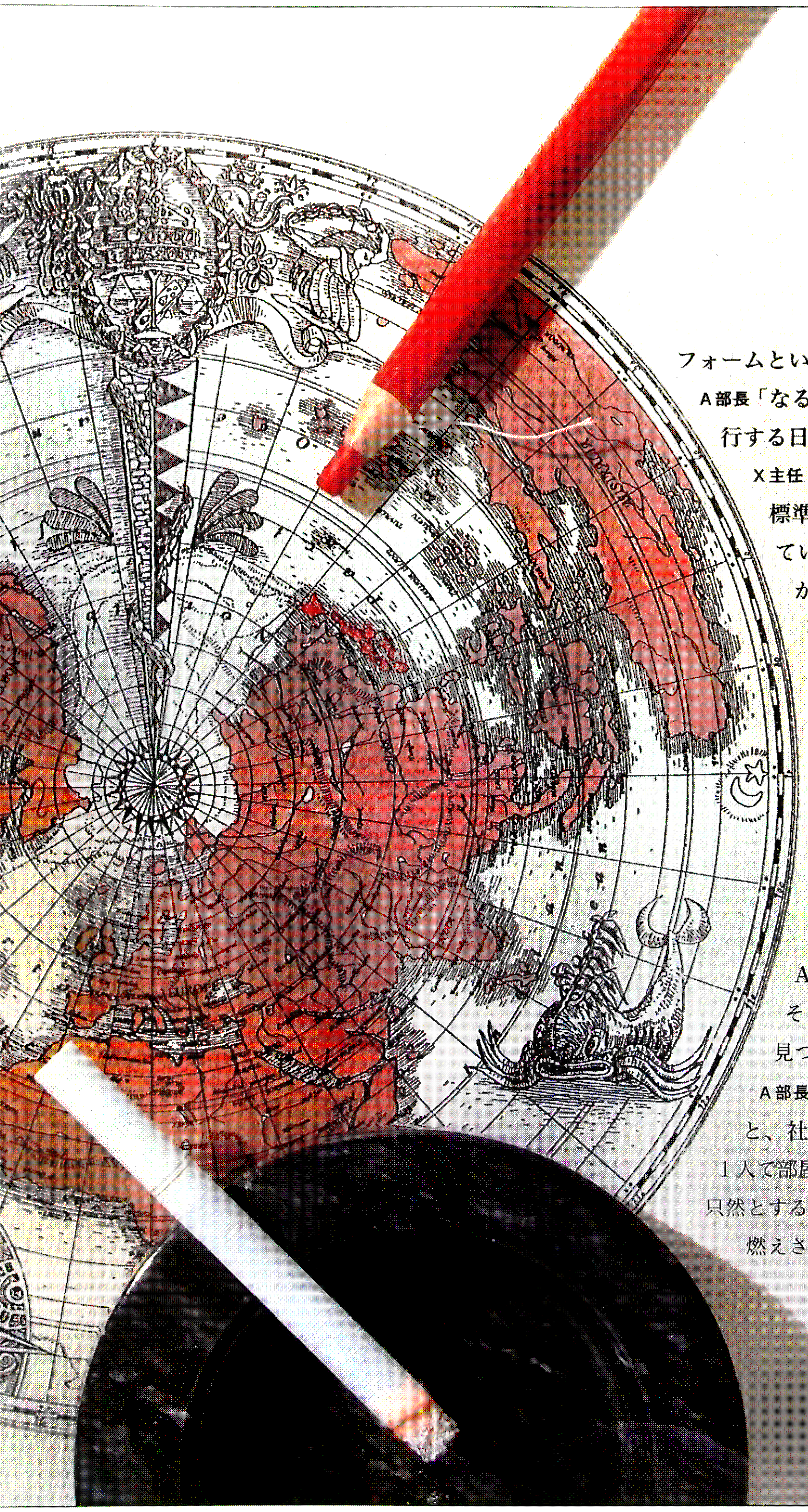
X主任「それにAXはVGAに対応していますからね。これらの画像は、640X480ドットの高品質な動画やアウトラインフォントを扱えるVGAモードでなくちゃ、世界のマルチメディアの流れにはついていけませんよ」

A部長、煙草を揉み消し、身を乗り出す。

X主任「オープンアーキテクチャのAXこそが、ネットワークコンピューティングを可能にする共通プラット

世界の標準機を、日本の標準機に。
我々が、パソコンの新しい時代を拓きます。

Acer　Canon
MITSUBISHI　NCR　OKI　olivetti　SANYO



THE ARCHITECTURE EXTENDED-AX

フォームといえるんです」

A部長「なるほど、日本市場が海外市場と同時進行する日も近いわけだ」

X主任「部長、事実上の世界標準機が、日本標準機になる日が、もうすぐそこまで来ているんです。新しい時代に備えて、今が決断の時です」

再び沈黙。A部長、14本目の煙草に火をつける。

その時ドアをノックする音がして、部長の秘書が現われる。

秘書「部長、会議のお時間です。もう社長がいらしてますので急いでください」

A部長「しまった、もうそんな時間か。すぐに行くと伝えてくれ」

A部長、慌てて立ち上がりドアに向かう。そこで振返って、X主任の目をまっすぐ見つめて言う。

A部長「よしわかった。……AXしかないと、社長に言うことにしよう」

1人で部屋に残されたX主任は信じられぬ思いで只然とする。が、すぐに満面に笑みを浮かべると、燃えさしの14本目の煙草を灰皿に押しつけた。

※MS-Windows 3.0は米国Microsoft社の登録商標です。
※IBM PC、IBM PC/AT、VGAは、米国IBM社の商標です。

CASIO　HEWLETT PACKARD　HITACHI　KYOCERA
SHARP　SONY　UNISYS

社名アルファベット順

お問合わせ：株式会社 翔泳社
TEL. 03-3263-0447

Brief

Full Screen Editor 日本語版

全米のプログラマが認めた実力!
アメリカでは「ブリーフ」が
プログラミング用エディターとして
最も多く利用されています。

C言語で組めるマクロ!

Version UP!

V.3.0 いよいよ登場。プログラミングエディター新世紀。

よりパワフルで、よりエレガントなプログラミングのために……。

●C言語で組めるマクロ 新機能

使い慣れているC言語でマクロが組めますから、習得が簡単でカスタマイズも思いのままです。

●マクロのスクリーン・デバッガ 新機能

ソースコードでデバックできます。もちろんブレークポイントやウォッチ・ドッグもセットできます。

●テンプレート機能〈構文の自動生成〉 新機能

C、Ada、Basic、COBOL、FORTRAN、MODULA2、Pascalで構文を自動生成します。

●300回ものアンドゥ

各ファイルごとに300までの操作を元にもどすことができますので、やり直しの面倒を気にせず、安心して編集できます。

●安全編集

Briefは、一定時間キーを操作しないこと最新の編集内容をファイルに自動保存します。だから編集中に電源が切れても安心です。

●その他の主な新機能

キー操作のファイルへのセーブとリプレイ、処理可能行数を拡張、リジューム機能、Ctrlキー・アサイン(オプション)、メニュー(オプション)etc。

アメリカでも日本でも高い評価

- ●プログラミング向きエディターを探している人は、もう探す必要はない。(Jerry Pournelle BYTE)
- ●ソフト開発には強力なツール、300回のアンドゥは大きな特徴。(日経バイト)
- ●既存のエディターに不満を覚えたら、これしかない。(Oh! FM)
- ●まるで夢のようなアンドゥだ――おもわず間違うのを期待してしまいそうだ。(The C Users Journal)

■主な機能

◆55ウィンドウ◆無限のファイル◆自動コンパイル◆自動エラー・ジャンプ◆自動インデント◆自動テンプレート◆バッファ書出 切換 削除 リスト表示◆ファイル名の指定 表示 変更◆バックアップ保存◆特殊文字入力◆2行を1行に連結◆ファイル単位の挿入◆増分検索◆各種マーク 始・終点の入換◆ブロックのカット コピー ペースト 印刷 保存 左右へのシフト◆対応する括弧へ移動◆C言語のコメント化 非コメント化◆DOSコマンドの実行◆キー操作記憶 実行◆入力ヒストリー◆英文テキスト整形◆タブのスペース挿入◆拡張子毎に異なった設定ができるタブ◆検索置換パターンの画面からの入力◆英字の大小文字変換 その他

定価 48,000円

(消費税は含まず)

■5機種対応パッケージ

(東芝、NEC、AX協議会、IBM、富士通)

●5"2DD、3.5"2DD デュアルメディア●MS-DOS 3.1以降に対応●英語DOS では使用できませんが、英語版 BRIEFもセット販売いたしております。●お急ぎの方、上記以外のメディア希望の方、英語BRIEF のセット販売希望の方はユーザー・サポート係までご連絡下さい。

日本システム株式会社

●お問合せは 東京都立川市曙町1-18-2一清ビル立川事業所Briefユーザー・サポート係
TEL0425-23-4469 FAX0425-27-6127

〈資料請求番号001〉

BERKELEY HACKERS バークレイ・ハッカーズ　新　刊

Zachary Margulis・Carol Palecki著　広谷渉訳　A5判228頁　定価1,800円

シリコンバレーの頭脳とも呼ばれるパソコンの聖地バークレイを中心に活躍するハッカー14人が半生を振り返る。『カッコウはコンピュータに卵を産む』の著述で一躍脚光を浴びた、クリフォード・ストール、いまアメリカのコンピュータ界で注目を集めるBSD UNIXのエキスパート、マイク・カーレルズをはじめ、これまであまり紹介されることのなかった彼らだが、その存在はコンピュータの歴史の中で見逃すことはできない。

GNU Emacs Lisp Manual Version18 2nd DRAFT　新　刊

Free Software Foundation編　A4判432頁　定価5,000円

本書は、1988年にBill Lewisを中心としてまとめられたEmacs Lisp Manual —— Emacs Version 18の第2版を翻訳したものです。オリジナルのEmacsに飽き足らない人、Emacsをもっと自由に使いこなしたい人にとって、Lispとこのマニュアルは不可欠です。

Illustrated C　近　刊

Leor Zolman著　池田憲昭訳　A5判350頁　予価3,800円

C言語を学習するには「ポインタ」と「ファイルI／O」という二つの難関を突破しなければなりません。著者は、C言語を構成する諸要素と他の要素や環境や設計方針との相互関係を、サンプルプログラムを用いることによって説明しています。本書は、Cの文法は勉強したがいま一つ「感じがつかめない」という方に最適な「中級用」の教科書です。この本を読むことによって、C言語についての多くの不満が解消されることでしょう。

98シリーズで使うVisual Basic入門　好評発売中

塚本　吉彦著　A5判208頁　定価980円

Windowsのプログラミングツールとして Microsoft社から発売されたVisual Basicは、ユーザインターフェイスに関する煩わしい作業をWindowsとVisual Basicが引き受けてくれますから、操作が比較的簡単です。またほかのプログラミングツールに比べて、39,800円と低価格であることも魅力の1つです。本書は現在販売されている英語版を、PC-9800シリーズの日本語環境で使うことを前提に書いてあります。

VZ倶楽部 (5"2HD付録ディスク付き)　好評発売中

ビレッジセンター編　A5判上製函入392頁　定価3,800円

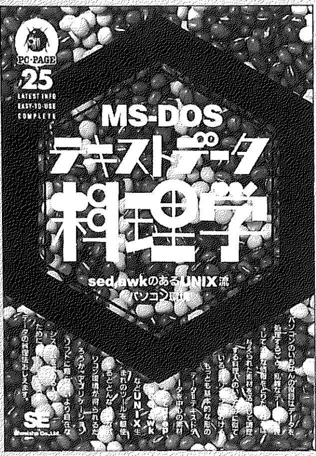
本書は、VZエディタのユーザたちによる合作です。八十余人のVZユーザが、それぞれ思いのままにつづった文章をこの一冊にまとめました。付録の5"2HDディスクには、VZエディタ Version 1.56の差分ファイルのほか、テキスト印刷ソフトPRT＋＋ Ver 5.22、EZエディタ Ver1.00を収録。

発売：株式会社ビレッジセンター出版局
〒101 東京都千代田区三崎町3-1-16 神田アメレックスビル2F
●TEL 営業部：03-3239-2187 編集部：03-3239-1950●FAX 03-3221-1768

定価はすべて税込みです

PC-PAGE どれをとっても栄養満点！おいしさいっぱいのパソコンブック

絶賛発売中



25 MS-DOS テキストデータ料理学

DOSのテキストデータを十二分に活用するためのガイドブック。

●定価1,500円・SE編集部編

24 MS-DOS5.0 全ファイル解説

98用MS-DOS5.0の全てのファイルの意味と利用法を解説。

●定価1,500円
SE編集部編

23 ハードディスクの達人

ビギナーでもハードディスクを120%活用できるマニュアル。

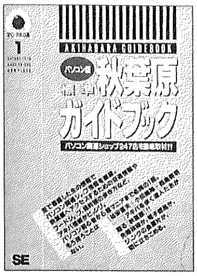
●定価1,500円
SE編集部編

22 パソコンのお医者さん

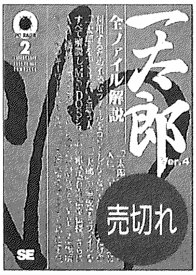
好評を博した「パソコン見栄講座」の著者による、パソコン版「家庭の医学」。

●定価1,500円
SE編集部編

SE Shoeisha Co., Ltd.　株式会社翔泳社　〒106東京都港区西麻布2-22-1スリーアップ西麻布　TEL.03-5467-0361

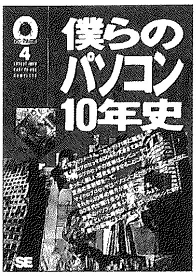


1 パソコン版標準秋葉原ガイドブック
定価1200円

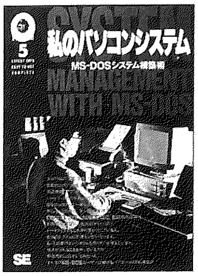


2 一太郎Ver.4全ファイル解説
定価1200円

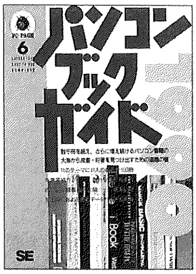
3 林檎百科
定価1200円



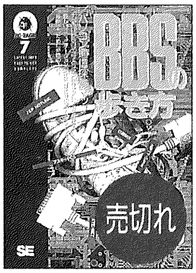
4 僕らのパソコン10年史
定価1200円



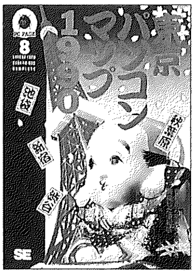
5 私のパソコンシステム
定価1200円



6 パソコンブックガイド1990
定価1200円

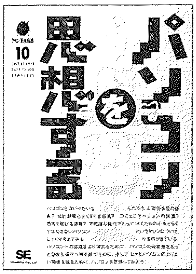


7 BBSの歩き方
定価1200円

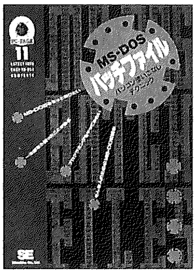


8 東京パソコンマップ1990
定価1200円

9 MS-DOS3.3全ファイル解説
定価1200円

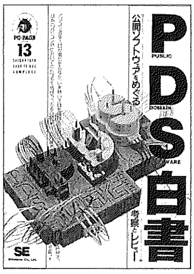


10 パソコンを思想する
定価1200円

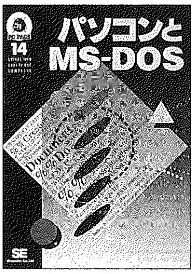


11 MS-DOSバッチファイル
定価1500円

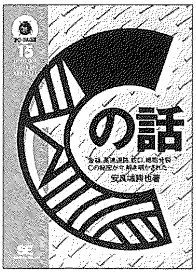
12 プロフェッショナルのパソコン利用術
定価1200円



13 PDS白書
定価1500円



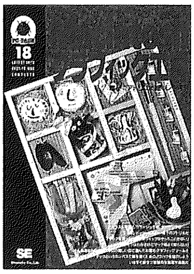
14 パソコンとMS-DOS
定価1200円



15 Cの話
定価1500円

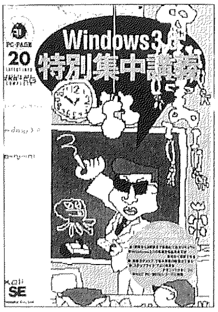
16 パソコン見栄講座
定価1500円

17 WINDOWSがやってきた
定価1200円



18 Macintoshグラフィック工房
定価1500円

19 日本語入力読本
定価1500円



20 Windows3.0特別集中講義
定価1500円

21 フリーソフトウェア派宣言
定価1500円

最新刊

NOW PRINTING

26
Mac派
DTP倶楽部
マックでミニメディアを作るための実践的解説書。

● 定価1,600円・SE編集部編

※定価はすべて税込価格となっております。

翔泳社の本は全国どの書店でもお求めになれます。店頭にない場合は「至急」とお申し出の上ご注文いただければ10日位でお届けできます。

Programmer's Page
MONTHLY MAGAZINE FOR PROGRAMMERS

特集

アセンブラの作り方

8086系簡易アセンブラの設計と作成　秋山登

1　アセンブリ言語と8086系CPUの機械コード　2
2　アセンブラ処理の概要と簡易アセンブラの処理方式　17
3　アセンブラの制御　19
4　入出力　28
5　シンボルの登録と参照　37
6　機械コードの生成　41
7　式　62

IC千夜一夜　第12回
CRTコントローラの系譜　虎岩登　69
実践C塾　アルゴリズム篇　第6回
ツリー・その2　加野島英渡・山本博昭　78
イカすBASICプログラミング入門　最終回
マイナーチェンジ編　酒井法雄　88
98活用アセンブラ技法　RS-232C編　第2回
RS-232Cドライバ　星野操　99

付録
5" 2HD フロッピーディスク

綴じ込み付録
8086インストラクション一覧

コンピュータのなかのSF　第3回
HAPPY BIRTHDAY HAL!　志賀隆生　74
プログラマの道具学　第18回
秋葉原　祐安重夫　76
PRODUCTS REVIEW
Microsoft QuickC for Windows　松尾治己　113
Book Review「アルゴリズム（原書第2版）」　中村正三郎　122
本のアンテナ　古田島義和
佐々木哲のU.S.プロダクツウォッチ　124
西川和久の希望的観測　126
コンピュータの法律　熊谷信太郎　128
アキバ短信
プログラマの仕事場　第15回　株式会社デジタル・リサーチ・ジャパン　130

INFORMATION PAGE　133
NEWS & NEW PRODUCTS　134
Programmer's Program　DRVR-55　138
オンラインソフト採集帳18「MAHALITO」　140
ドラゴン・ポスト　142
付録ディスクのご案内　144
バーチャルスペースガイド　145
読者プレゼント
クイズ・ランダムスクリーン　146
SE-NETからのお知らせ
バックナンバーのご案内　147

広告索引　資料請求番号

AX メーカー連合広告
日本システム(株)…………001
ビレッジセンター
(財)新世代コンピュータ技術開発機構
T•ZONE…………002
(株)ライフボート…………P03
ボーランド(株)…………P04

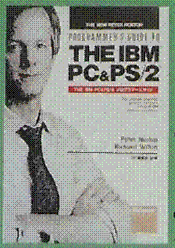
人材募集広告

(株)エルテックス

1992 JUNE No.18

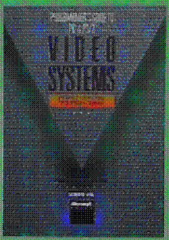
翔泳社が贈るプログラマ必携書

IBM PC & PS/2 プログラミング 3部作



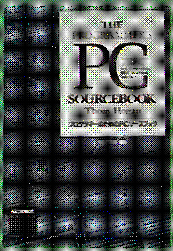
あのピーター・ノートンのベストセラー、IBMプログラマ必携のスタンダードガイダンス。

IBM PC & PS/2
プログラマーズガイド
● Peter Norton & Richard Wilton著
●定価　5,200円



IBM PCファミリーのビデオシステム完全ガイド。

PC & PS/2 ビデオシステム
プログラマーズガイド
● Richard Wilton著
●定価　6,000円



IBM PC & PS/2プログラマ必携の技術資料完訳。

プログラマーのための
PC ソースブック
● Thom Hogan著
●定価　5,200円

FGCS'92

21世紀のコンピュータ

第五世代コンピュータ国際会議1992

INTERNATIONAL CONFERENCE ON FIFTH GENERATION COMPUTER SYSTEMS 1992

1992年6月1日(月)〜6月5日(金)　東京プリンスホテル

第五世代コンピュータ国際会議1992『FGCS'92』が'81、'84、'88に続いて開催。重要な国際会議として定着したFGCS、世界初の論理型プログラミングによる超並列推論コンピュータのデモンストレーションをはじめとする研究成果の発表・意見交換の場として高く評価され、一流の研究者(デンマーク工科大学D.Bjørner教授、シラキュース大学J.A.Robinson教授、オックスフォード大学C.A.R.Hoare教授、インペリアルカレッジRobert Kowalski教授など)、技術者や政府関係者が世界中から参集。多くの皆様の積極的なご参加を念願します。

開催要項

主　催

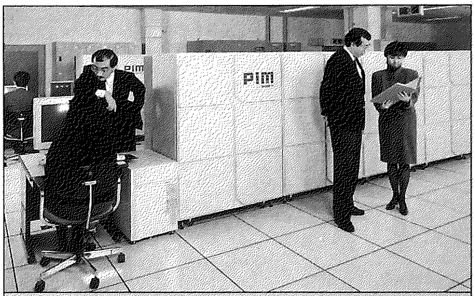
(財)新世代コンピュータ技術開発機構(ICOT)

社交行事

- 歓迎レセプション(無料)：6月1日(月)
  午後6時30分
- バンケット(¥25,000/1名)：6月4日(木)
  午後7時
- バンケットスピーチ：唐津　一(東海大学教授)

登録費

¥110,000(一般)
¥10,000(学生)
登録費には予稿集代(学生は除く)と消費税が含まれます。



PIM/m。256個の推論プロセッサを2次元のメッシュ・ネットワークによって接続し、100M LIPS以上の性能を達成。

会議日程　1992年6月1日(月)〜5日(金)

	6月1日(月)	6月2日(火)	6月3日(水)	6月4日(木)	6月5日(金)
	全体会議		テクニカルセッション		
9:30 午前	あいさつ 祝辞 基調講演 報告	招待講演 J.A.Robinson (米、シラキュース大学教授) ICOT研究成果報告	招待論文及び一般論文の発表(下記4セッション並行) 基礎理論 招待論文：Raymond Reiter(カナダ、トロント大学教授)		
午後	招待講演 D.Bjørner (デンマーク工科大学教授) ICOT研究開発総合報告 FGCSプロジェクト10年間の概観 FGCSプロジェクトにおける基礎研究の展開 並列推論マシンと基本ソフトウェアの研究開発経緯	並列推論マシンPIM オペレーティングシステムPIMOS 知識ベース管理システム 制約論理プログラミングシステム 並列定理証明システムとその応用 自然言語処理ソフトウェア 並列応用実験ソフトウェア	アーキテクチャとソフトウェア 招待論文：William Dally(米、MIT教授) アプリケーションと社会的インパクト 招待論文：Ivan Bratko(ユーゴスラビア ヨゼフ・ステファン研究所教授) ICOT研究トピックス		招待講演 C.A.R.Hoare (英、オックスフォード大学教授) パネル討論 21世紀の情報処理への出発点 座長：Robert Kowalski (英、インペリアルカレッジ教授) 閉会あいさつ ICOT専務理事
5:30 夕	歓迎レセプション			バンケット	
	デモンストレーション(午後)	デモンストレーション(午前、午後)			

事務局

〒108 東京都港区三田1-4-28 三田国際ビル21階 (財)新世代コンピュータ技術開発機構(ICOT) FGCS'92 事務局
電話(03)3456-3195 FAX(03)3456-3158 E-mail:fgcs'92@icot.or.jp

ASSEMBLER

特集

アセンブラの作り方

8086系簡易アセンブラの設計と作成

秋山登

いまどきアセンブラを作ることなどあまり流行らないかもしれない．しかし，自分でアセンブラを作らないまでもアセンブラの中身をのぞいてみると，結構アセンブラに強くなるものである．MS-DOSの世界には，MASMを代表とする高級なアセンブラが数多くあるが，これらの中身は簡単にのぞくわけにはいかない．そこで，筆者が勉強がてら作成した486までの命令が処理できる8086系の簡易アセンブラを紹介することにする．

ASSEMBLER

特集 アセンブラの作り方

1 アセンブリ言語と8086系CPUの機械コード

アセンブラは，ソースプログラムを機械コード（機械語）に変換するプログラムであるから，機械コードの構造に大きく依存する．そこで，プログラムの説明に入る前に製作した簡易アセンブラに関係する8086系の機械コードとアセンブリ言語の対応をおさらいを兼ねて説明しておこう．なお，製作したアセンブラは，MASMのアセンブリ言語のサブセット版と考えて差し支えない．

1-1 アセンブリ言語の要素と書式

アセンブリ言語の要素は，シンボル，命令，オペランド，およびコメントの4つである．これらの要素は，1つの行の中で次のような順序で記述することができる．

【シンボル】【命令およびオペランド】【;コメント】

この書式で，【】は省略可能であることを示すもので，実際のプログラムの中にはもちろん記述できない．各要素と要素の間は，文法上必要とされるコロン(:)，セミコロン(;)およびコンマ(,)のある場合を除いて1個以上のスペースかTABで区切られる．シンボルの前にスペースやTABを置くことも許される．

シンボルには，3つの種類がある．命令の中で，とくにプロセッサ命令と呼ばれる機械コードを生成する命令につけたシンボルをラベルという．ラベルの場合は，通例シンボルの最後にコロン(:)がおかれる．命令の中でディレクティブ（宣言·疑似命令などともいう）と呼ばれるデータや作業領域を確保する命令につけたシンボルは，バリアブル（変数）である．また，シンボルと値を結び付けるディレクティブによって直接的な数値に結び付けられたシンボルは，ナンバーと呼ばれる．通常のアセンブラでは，シンボルをラベルと呼ぶことが多いが，このように8086系のアセンブラではラベルに特別な意味があるので，以下の文中では混乱を避けるためにシンボルと呼ぶことにする．

シンボルには，ある程度の制限はあるが，記憶しやすい任意の単語や略語を記述することができる．とくにラベルとバリアブルは，命令によって機械コードやデータが置かれる最初のアドレスを表すようアセンブラが処理してくれる．したがって，機械コードによるプログラミングのようにユーザはアドレスを計算しなくても済む．また，ナンバーは入出力ポートのアドレスや，VRAMのアドレスといった固定的な数値やアドレスを記憶しやすい単語に置き換える．

命令には，前記のように2つの種類がある．プロセッサ命令とディレクティブである．プロセッサ命令とディレクティブには，必ずアセンブラによって決まっているニモニックと呼ばれる略語を記述しなければならない（本文では述べないがマクロ命令の呼び出しは例外）．プロセッサ命令は，次に述べるオペランドとともにCPUによって直接実行される機械コードに変換される．プロセッサ命令のニモニックは，

アセンブラ用語集
アセンブラに関する用語を集めてみました．

プロセッサのメーカーが公表しているニモニックが使用される．8086系CPUのアセンブラでは，プリフィックス命令と呼ばれる特殊なプロセッサ命令を通常のプロセッサ命令の前に置くこともできる．ディレクティブは，アセンブラを制御したり，データを確保したり，シンボルに強制的に値を与えたりするのに使用される．ディレクティブは，同じCPUを対象としたアセンブラでも共通なものとそうでないものが存在するのが普通である．

オペランドは，命令の対象を指定する．たとえば，XにYを加算するという命令のXとYはオペランドである．オペランドは，コンマ(,)で区切って複数個書くことができる．しかし，オペランドの個数と内容は，書くことができるかできないかも含めて，命令によって決まっている．命令を書いたときには，その命令が必要とするオペランドは必ず書かなければならない．逆に命令がないときやオペランドが不要な命令を書いたときには書くことはできない．これも8086系のアセンブラに特有なことだが，セグメントオーバーライドと呼ばれるプロセッサ命令は，このオペランドの中に現れる．

コメントは，C言語の/*と*/に囲まれたコメントと同様で，何が書いてあってもアセンブラに無視される要素である．コメントはプログラムをコーディングしたプログラマ自身やプログラムを読む人に対するプログラムの説明に使用される．アセンブラのコメントは，セミコロン(;)から始まり行の最後まで続くのでC言語のように任意の場所にコメントを書くことはできない．

以上をまとめると次のプログラム例のようになる（例1）．

例1

```
 1:  ; 画面を消去する
 2:
 3:  VRAMSEG     EQU    0A000H
 4:  MOJISUU     EQU    2000
 5:
 6:  CLRSCRN:    MOV    AX,VRAMSEG
 7:              MOV    ES,AX
 8:              MOV    AX,CS:CLRDAT
 9:              MOV    CX,MOJISUU
10:              XOR    DI,DI
11:              CLD
12:              REP    STOSW   ; 2BYTEが1文字
13:              RET            ; 処理終了
14:  CLRDAT      DW     0020H   ; 消去データ
```

例1で，第1行目はコメントだけの行である．2行目と5行目はすべての要素が省略された行である．3行目と4行目にはシンボルが付けられている．これらのシンボルはEQUというディレクティブによって，それぞれ直接的な数値である16進数のA000と10進数の2000になるよう指定されている．そこでVRAMSEGとMOJISUUはナンバーとなる．0A000Hと2000は，EQUディレクティブのオペランドである．

6行目から12行目までに書かれているMOV，XOR，CLD，REP，STOSWおよびRETは，プロセッサ命令である．6行目のCLRSCRNというシンボルは，プロセッサ命令に付けられているのでラベルであり，ラベルに必要なコロン(:)が後ろに書かれている．MOVおよびXORは，2つのオペランドが必要な命令である．AX，VRAMSEG，ES，CS:CLRDAT，CX，MOJISUU，およびDIは，これらの命令のオペランドである．CLDとRETは，オペランドの不要な命令であり，オペランドは書かれていない．REPはプリフィックス命令であり，その後にSTOSWというオペランドの不要な命令が書かれている．もしオペランドの必要な命令が来るときには，さらにこの後にオペランドを書くことになる，12行目と13行目の最後にはコメントが記述されている．

14行目は，16ビットの16進数0020というデータをRETの命令のあるメモリの直後に記憶させるディレクティブであるから，これに付けられたシンボルCLRDATはバリアブルである．0020HはDWディレクティブのオペランドである．この行の最後にもコメントが書かれている．

1-2 アドレッシングモード

アセンブラのオペランドには，レジスタやメモリの特定の位置を指定するために，いろいろな方法を指定することができる．これらの指定方法をアドレッシングモードと呼んでいる．8086系CPUでは，次のようなアドレッシングモードが使用できる．

1-2-1 インプライド

おもにオペランドの不要な命令で多

アセンブラ用語集
ANK　Alphabet（英字）・Number（数字）・Kana（カタカナ）を意味し，8ビットで表現できる文字セットのこと．

く見られるように，命令自身にオペランドが含まれているアドレッシングモードである．たとえば，CLIやSTDなどは，フラッグレジスタの特定のビットをオペランドとする．

1-2-2 レジスタ直接

オペランドをレジスタとする命令である．レジスタの内容が命令の操作の対象になる．演算命令やデータ移動命令など，多くの命令でオペランドとして使用できる．たとえば，

```
ADD     SI,AX
PUSH    DX
CALL    SI
```

などは，すべてオペランドがレジスタ直接である．

1-2-3 イミーディエット

命令自身に組み込まれた直接数値をオペランドとする．たとえば，

```
PUSH    -1
ENTER   COUNT,0
```

などのオペランドがイミーディエットオペランドである．アセンブラは，COUNTのようなシンボルが記述されると，シンボルがナンバーであるときイミーディエットオペランドとして処理する．イミーディエットアドレッシングモードをオペランドにとる命令には，イミーディエットオペランドが符号付きの1バイトの数値に収まるデータのとき，短い形式の命令を使用できるものがある．

1-2-4 IP相対

IP相対のアドレッシングモードを持つ命令は，その命令の次にある命令の先頭のアドレスから何バイト後か先かを示すオペランドをとる．次の例のようにアセンブラではイミーディエットアドレスモードや直接アドレスモードと書式上区別がつかない．

```
CALL    CLEAR
JMP     OWARI
```

このほかの命令では，条件付きジャンプ命令や，LOOP命令などで，このアドレッシングモードを使用することができる．これらの命令では，オペランドにラベルがくるとアセンブラがIP相対モードであると判断する．

1-2-5 直接アドレス

バリアブルや命令のオフセットアドレスや，ポインタをそのままオペランドに指定するアドレッシングモードである．たとえば，

```
DEC     DATACOUNT
CALL    FAR  PTR  DATA-
SHORI
```

などのオペランドは，直接アドレスのアドレッシングモードである．バリアブルのオフセットアドレスの指定は，上記DEC命令のほか，多くの命令で可能であるが，ポインタをオペランドに指定できる命令は，CALLとJMPの2つの命令だけである．アセンブラは，オペランドがバリアブルかポインタかによって，直接アドレスのアドレッシングモードであることを識別する．

1-2-6 レジスタ修飾

レジスタ修飾アドレッシングモードには次の10のタイプがある．

タイプ1
  [bsr]および[ixr]
タイプ2
  d8[bsr]およびd8[ixr]
タイプ3
  d16[bsr]およびd16[ixr]
タイプ4
  d32[bsr]
タイプ5
  [bsr+ixr]
タイプ6
  d8[bsr+ixr]
タイプ7
  d16[bsr+ixr]
タイプ8
  [bsr+ixr * s]
タイプ9
  d8[bsr+ixr * s]
タイプ10
  d32[bsr+ixr * s]

上の記号のうち，bsrはベースレジスタ，ixrはインデックスレジスタ，d8は8ビットの符号つき変位（displacement），d16は16ビットの符号つき変位，d32は32ビットの符号つき変位，sはスケールである．タイプ1はレジスタ間接アドレッシングモードとも呼ばれ，レジスタの内容がオペランドのオフセットアドレスとなる．タイプ2からタイプ7までは，bsrやixrの内容と変位をすべて加算した結果がオペランドのオフセットとなるアドレッシン

アセンブラ用語集
BCD　Binary Coded Decimal（2進化10進）の略．桁数が大きくなると本来の10進数とは多少の誤差が生じる．

グモードで，インデックス修飾とも呼ばれる．タイプ8，9および10は，スケールドインデックスと呼ばれる．この2つのタイプでは，ixrの内容をs倍してその結果に変位とbsrの内容を加算した結果がオペランドのオフセットアドレスとなる．sには1，2，4，8の4つの数値が指定できる．

16ビットのアドレスサイズでは，タイプ1，2，3，5，6，7のみが使用できる．このとき，bsrにはBXとBPのレジスタが使用でき，ixrにはSIとDIレジスタが使用できるのみである．

32ビットのアドレスサイズでは，タイプ1，2，4，6，8，9のみが使用できる．ただし，タイプ8と9にs＝1を指定したときは，それぞれタイプ5と6と同じである．bsrには，32ビットの汎用レジスタすべて（EAX，EBX，ECX，EDX，ESI，EDI，EBP，ESP）が指定できる．ixrには，ESPを除く32ビットの汎用レジスタが指定できる．

レジスタ修飾をアセンブラのプログラムに書くときは，上にあげた書式以外にいろいろな書式が可能である．たとえばタイプ6では，

```
[bsr][ixr][d8]
[bsr+ixr+d8]
[bsr+d8][ixr]
[bsr][ixr+d8]
```

などはいずれも同じタイプ6を表している．次の例は，レジスタ修飾アドレッシングモードの使用例である．

```
MOV   AX,[EBX]
MOV   10[EBP+EDI＊2],AX
MOV   BX,DT1[SI]
MOV   CX,DT2[BX+DI]
```

変位は，バリアブルまたはナンバーが使用できるが，変位のないときや変位がナンバーであるときは，オペランドのサイズをPTR演算子を使用して明示しなければならない場合がある．

1-3 プロセッサ命令とオペランドのタイプ

8086系CPUには，インテル社の8086/186/286/386/486の5つのタイプのCPUがある．これら5つのタイプのCPUのすべてのプロセッサ命令を必要とするオペランドの数別にまとめたのが表1から表4である．ただし，NDP（コプロセッサ）の命令は今回サポートしないので除外してある．これらの表を作る作業は，チップのマニュアルや参考書を片手に根気と労力のいる仕事である．しかし，これらの表をうまく作ることができれば，どのようなCPUに対するアセンブラでも，アセンブラを製作する半分の作業は終わったようなものである．

表の最初の欄は，アセンブリ言語の命令に使用するニモニック（命令の略称）である．これには説明は不要であろう．ただ，表1の中のASIZEとOSIZEは仮にこのようなニモニックを付けたが，標準的なアセンブラでは命令としては使用できず，アセンブラがニモニックやオペランドから自動的に判断して生成する命令である．この2つの命令は，386以上のCPUでアドレスサイズとオペランドサイズを16ビットと32ビットの間で切り替える命令である．詳細は，「1-4」で説明する．

表2～表4のオペランドの必要な命令では，オペランドに書くことのできる内容を略記している．次にその記号の意味を説明しておこう．

(A)/

/の前の内容は16ビットサイズの動作時の内容であり，/の後の内容は32ビットサイズ動作時の内容であることを示す．CPUの欄に8086とあっても286以下のCPUでは，32ビットサイズ動作時の内容は無効である．

(B)1，3の単独の数字

オペランドの値が1や3のとき命令が有効であることを示す．

(C)AL，AX，EAX，CL，DX，CRn，DRn，TRn

オペランドにこれらのレジスタが指定されたときのみ命令が有効であることを示す．nは0から7の範囲のレジスタ番号である．

(D)d8 d16 d32

IP相対オペランドを示す．8，16および32は，IPからの相対値として許されるオフセットのサイズを示すと同時に機械コードに占めるビット長を表す．

(E)i(6) i(8) i(16) i(32)

イミーディエットオペランドを示す．6，8，16および32の数字は機械コードに占めるイミーディエットデータのビット長である．

(F)is(8)

符号つき8ビット長のイミーディエットオペランドを示す．

(G)l16 l32 l48

直接アドレスのオペランドを示す．16，32および48は直接アドレスの値として許される値の範囲を示すと同時に機械コードに占めるビット長を表す．

アセンブラ用語集
BIOS　Basic Input and Output System（基本入出力システム）の略で，システムコールで呼び出すことができる．

(H)m

直接アドレスおよびレジスタ修飾のオペランドのいずれかであることを示す．オペランドのサイズはどのようなサイズでも構わない．直接アドレスの場合は値によって，レジスタ修飾の場合は，使用するレジスタによってアドレスサイズが決定される．

(I)m(D) m(F) m(Q)

直接アドレスまたはレジスタ修飾のオペランドであることを示す．()内のD, FおよびQは，オペランドのサイズがそれぞれ2ワード，3ワードおよび4ワードであることを示す．直接アドレスの場合は値によって，レジスタ修飾の場合は，使用するレジスタによってアドレスサイズが決定される．

(J)mr(W) mr(D)

レジスタ直接，直接アドレスまたはレジスタ修飾のオペランドであることを示す．()内のB, W, Dは，それぞれオペランドのサイズがバイト，ワード，および2ワードであることを示す．直接アドレスの場合は値によって，レジスタ修飾の場合は，使用するレジスタによってアドレスサイズが決定される．

(K)r(B) r(W) r(D)

レジスタ直接オペランドを示す．()の中のB, W, Dはそれぞれバイト，ワードおよび32ビットの汎用レジスタであることを示すと同時にオペランドサイズも示している．

(L)s2 s3

s2はCS, DS, ES, SSのセグメントレジスタを，S3はCS, DS, ES, SS, FSおよびGSのセグメントレジスタを示す．ただし，286以下のCPU

表1 オペランドの不要なプロセッサ命令

ニモニック	機械コード	CPU
AAA	37	8086
AAD	D5 0A	8086
AAM	D4 0A	8086
AAS	3F	8086
ASIZE	67	386
CBW	$$ 98	8086
CDQ	## 99	386
CLC	F8	8086
CLD	FC	8086
CLI	FA	8086
CLTS	0F 06	286
CMC	F5	8086
CMPSB	A6	8086
CMPSD	## A7	386
CMPSW	$$ A7	8086
CS:	2E	8086
CWD	$$ 99	8086
CWDE	## 98	386
DAA	27	8086
DAS	2F	8086
DS:	3E	8086
ES:	26	8086
FS:	64	386
GS:	65	386
HLT	F4	8086
INSB	6C	186
INSD	## 6D	386
INSW	$$ 6D	186
INTO	CE	8086
INVD	0F 08	486
IRET	$$ CF	8086
IRETD	## CF	386
LAHF	9F	8086
LEAVE	C9	186
LOCK	F0	8086

ニモニック	機械コード	CPU
LODSB	AC	386
LODSD	## AD	386
LODSW	$$ AD	8086
MOVSB	A4	8086
MOVSD	## A5	386
MOVSW	$$ A5	8086
NOP	90	8086
OSIZE	66	386
OUTSB	6E	186
OUTSD	## 6F	386
OUTSW	$$ 6F	186
POPA	$$ 61	186
POPAD	## 61	386
POPF	$$ 9D	8086
POPFD	## 9D	386
PUSHA	$$ 60	186
PUSHAD	## 60	386
PUSHF	$$ 9C	8086
PUSHFD	## 9C	386
REP	F3	8086
REPE	F3	8086
REPNE	F2	8086
REPNZ	F2	8086
REPZ	F3	8086
SAHF	9E	8086
SCASB	AE	8086
SCASD	## AF	386
SCASW	$$ AF	8086
SS:	36	8086
STC	F9	8086
STD	FD	8086
STI	FB	8086
STOSB	AA	8086
STOSD	## AB	386
STOSW	$$ AB	8086
WAIT	9B	8086
WBINVD	0F 09	486

注：記号の説明は「1-3」を参照

アセンブラ用語集

NEAR型　分岐命令やサブルーチンコールでジャンプ先が同一セグメント（64KB）内にある場合を指す．セグメント外にジャンプ先がある場合はFAR型という．

表2　オペランドを1つ必要とするプロセッサ命令

ニモニック	第1OPD	機械コード	CPU
BSWAP	r(D)	## 0F C8+r	486
CALL	d16/32	E8 d16/32	8086
CALL	mr(W/D)	FF 10+mr ??	8086
CALL	I32/48	9A I32/48	8086
CALL	m(D/F)	FF 18+m ??	8086
DEC	mr(B)	FE 08+mr ??	8086
DEC	mr(W/D)	FF 08+mr ??	8086
DEC	r(W/D)	48+r	8086
DIV	mr(B)	F6 30+mr ??	8086
DIV	mr(W/D)	F7 30+mr ??	8086
IDIV	mr(B)	F6 38+mr ??	8086
IDIV	mr(W/D)	F7 38+mr ??	8086
INC	mr(B)	FE 00+mr ??	8086
INC	mr(W/D)	FF 00+mr ??	8086
INC	r(W/D)	40+r	8086
INT	3	CC	8086
INT	i(8)	CD i(8)	8086
INVLPG	m	0F 01 38+m ??	486
JA	d8	77 d8	8086
JA	d16/32	0F 87 d16/32	386
JAE	d8	73 d8	8086
JAE	d16/32	0F 83 d16/32	386
JB	d8	72 d8	8086
JB	d16/32	0F 82 d16/32	386
JBE	d8	76 d8	8086
JBE	d16/32	0F 86 d16/32	386
JC	d8	72 d8	8086
JC	d16/32	0F 82 d16/32	386
JCXZ	d8	&& E3 d8	8086
JECXZ	d8	%% E3 d8	386
JE	d8	74 d8	8086
JE	d16/32	0F 84 d16/32	386
JG	d8	7F d8	8086
JG	d16/32	0F 8F d16/32	386
JGE	d8	7D d8	8086
JGE	d16/32	0F 8D d16/32	386
JL	d8	7C d8	8086
JL	d16/32	0F 8C d16/32	386
JLE	d8	7E d8	8086
JLE	d16/32	0F 8E d16/32	386
JMP	d16/32	E9 d16/32	8086
JMP	mr(W/D)	FF 20+mr ??	8086
JMP	I32/48	EA I32/48	8086
JMP	m(D/F)	FF 28+m ??	8086
JMP	d8	EB d8	8086
JNA	d8	76 d8	8086
JNA	d16/32	0F 86 d16/32	386
JNAE	d8	72 d8	8086
JNAE	d16/32	0F 82 d16/32	386
JNB	d8	73 d8	8086
JNB	d16/32	0F 83 d16/32	386
JNBE	d8	77 d8	8086
JNBE	d16/32	0F 87 d16/32	386
JNC	d8	73 d8	8086
JNC	d16/32	0F 83 d16/32	386
JNE	d8	75 d8	8086
JNE	d16/32	0F 85 d16/32	386
JNG	d8	7E d8	8086
JNG	d16/32	0F 8E d16/32	386
JNGE	d8	7C d8	8086
JNGE	d16/32	0F 8C d16/32	386
JNL	d8	7D d8	8086
JNL	d16/32	0F 8D d16/32	386
JNLE	d8	7F d8	8086
JNLE	d16/32	0F 8F d16/32	386
JNO	d8	71 d8	8086
JNO	d16/32	0F 81 d16/32	386
JNP	d8	7B d8	8086
JNP	d16/32	0F 8B d16/32	386
JNS	d8	79 d8	8086
JNS	d16/32	0F 89 d16/32	386
JNZ	d8	75 d8	8086
JNZ	d16/32	0F 85 d16/32	386
JO	d8	70 d8	8086
JO	d16/32	0F 80 d16/32	386
JP	d8	7A d8	8086
JP	d16/32	0F 8A d16/32	386
JPE	d8	7A d8	8086
JPE	d16/32	0F 8A d16/32	386
JPO	d8	7B d8	8086
JPO	d16/32	0F 8B d16/32	386
JS	d8	78 d8	8086
JS	d16/32	0F 88 d16/32	386
JZ	d8	74 d8	8086

アセンブラ用語集
アキュムレータ　汎用レジスタの中でもAXレジスタを指しアキュムレータという。算術、論理演算の中心となるレジスタ。

ニモニック	第1OPD	機械コード	CPU
JZ	d16/32	0F 84 d16/32	386
LGDT	m(F)	0F 01 10+m ??	286
LIDT	m(F)	0F 01 18+m ??	286
LLDT	mr(W)	0F 00 10+mr ??	286
LMSW	mr(W)	0F 01 30+mr ??	286
LODS	mr(B)	AC	8086
LODS	mr(D)	## AD	386
LODS	mr(W)	$$ AD	8086
LOOP	d8	E2 d8	8086
LOOPE	d8	E1 d8	8086
LOOPNE	d8	E0 d8	8086
LOOPNZ	d8	E0 d8	8086
LOOPZ	d8	E1 d8	8086
LTR	mr(W)	0F 00 18+mr ??	286
MUL	mr(B)	F6 20+mr ??	8086
MUL	mr(W/D)	F7 20+mr ??	8086
NEG	mr(B)	F6 18+mr ??	8086
NEG	mr(W/D)	F7 18+mr ??	8086
NOT	mr(B)	F6 10+mr ??	8086
NOT	mr(W/D)	F7 10+mr ??	8086
POP	mr(W/D)	8F 00+mr ??	8086
POP	r(W/D)	58+r	8086
POP	s2	07+s2＊8	8086
POP	FS	0F A1	386
POP	GS	0F A9	386
PUSH	mr(W/D)	FF 30+mr ??	8086
PUSH	r(W/D)	50+r	8086
PUSH	s2	06+s2＊8	8086
PUSH	FS	0F A0	386
PUSH	GS	0F A8	386
PUSH	is(8)	6A is(8)	186
PUSH	i(16/32)	68 i(16/32)	186
RET		C3	8086
RET	i(16)	C2 i(16)	8086
RETF		CB	8086
RETF	i(16)	CA i(16)	8086
SCAS	mr(B)	AE	8086
SCAS	mr(D)	## AF	386
SCAS	mr(W)	$$ AF	8086
SETA	mr(B)	0F 97 00+mr ??	386
SETAE	mr(B)	0F 93 00+mr ??	386

ニモニック	第1OPD	機械コード	CPU
SETB	mr(B)	0F 92 00+mr ??	386
SETBE	mr(B)	0F 96 00+mr ??	386
SETC	mr(B)	0F 92 00+mr ??	386
SETE	mr(B)	0F 94 00+mr ??	386
SETG	mr(B)	0F 9F 00+mr ??	386
SETGE	mr(B)	0F 9D 00+mr ??	386
SETL	mr(B)	0F 9C 00+mr ??	386
SETLE	mr(B)	0F 9E 00+mr ??	386
SETNA	mr(B)	0F 96 00+mr ??	386
SETNAE	mr(B)	0F 92 00+mr ??	386
SETNB	mr(B)	0F 93 00+mr ??	386
SETNBE	mr(B)	0F 97 00+mr ??	386
SETNC	mr(B)	0F 93 00+mr ??	386
SETNE	mr(B)	0F 95 00+mr ??	386
SETNG	mr(B)	0F 9E 00+mr ??	386
SETNGE	mr(B)	0F 9C 00+mr ??	386
SETNL	mr(B)	0F 9D 00+mr ??	386
SETNLE	mr(B)	0F 9F 00+mr ??	386
SETNO	mr(B)	0F 91 00+mr ??	386
SETNP	mr(B)	0F 9B 00+mr ??	386
SETNS	mr(B)	0F 99 00+mr ??	386
SETNZ	mr(B)	0F 95 00+mr ??	386
SETO	mr(B)	0F 90 00+mr ??	386
SETP	mr(B)	0F 9A 00+mr ??	386
SETPE	mr(B)	0F 9A 00+mr ??	386
SETPO	mr(B)	0F 9B 00+mr ??	386
SETS	mr(B)	0F 98 00+mr ??	386
SETZ	mr(B)	0F 94 00+mr ??	386
SGDT	m(F)	0F 01 00+m ??	286
SIDT	m(F)	0F 01 08+m ??	286
SLDT	mr(W)	0F 00 00+mr ??	286
SMSW	mr(W)	0F 01 20+mr ??	286
STOS	mr(B)	AA	8086
STOS	mr(D)	## AB	386
STOS	mr(W)	$$ AB	8086
STR	mr(W)	0F 00 08+mr ??	286
VERR	mr(W)	0F 00 20+mr ??	286
VERW	mr(W)	0F 00 28+mr ??	286
XLAT		D7	8086
XLAT	mr	D7	8086

注：記号の説明は「1-3」を参照

アセンブラ用語集
アドレッシングモード　扱うメモリアドレスの参照方法をいう．アドレスをレジスタを介さずオペランドで直接参照する場合を直接アドレッシングという．

では，FSとGSは無効である．

機械コードの欄は，機械コードとオペランドの対応を記号化して示している．??の部分とd16/32, l32/48やi(16/32）のようにビット長を示している部分を除いて，スペースで区切っている部分が1バイトの切れ目である．数字あるいは大文字で書かれている場所は，変化しない機械コードで命令コードとも呼ばれる．次にオペランドと同様に記号の意味を説明しよう．

(M)##

16ビットサイズで動作時には，オペランドサイズのプリフィックス命令が必要であることを示す．

(N)$$

32ビットサイズで動作時には，オペランドサイズのプリフィックス命令が必要であることを示す．

(O)%%

16ビットサイズで動作時には，アドレスサイズのプリフィックス命令が必要であることを示す．

(P)&&

32ビットサイズで動作時には，アドレスサイズのプリフィックス命令が必要であることを示す．

(Q)??

レジスタ修飾では変位が，直接アドレスモードではオフセットが，この位置を占めることを示す．

(R)mmr

レジスタ直接とレジスタ修飾あるいは直接アドレスの2つのオペランドをコード化して指定するバイトである．

表3　オペランドを2つ必要とするプロセッサ命令

ニモニック	第1OPD	第2OPD	機械コード	CPU
ADC	AL	i(8)	14 i(8)	8086
ADC	AX/EAX	i(16/32)	15 i(16/32)	8086
ADC	mr(B)	i(8)	80 10+mr ?? i(8)	8086
ADC	mr(W/D)	is(8)	83 10+mr ?? is(8)	8086
ADC	mr(W/D)	i(16/32)	81 10+mr ?? i(16/32)	8086
ADC	mr(B)	r(B)	10 mrr ??	8086
ADC	mr(W/D)	r(W/D)	11 mrr ??	8086
ADC	r(B)	mr(B)	12 mrr ??	8086
ADC	r(W/D)	mr(W/D)	13 mrr ??	8086
ADD	AL	i(8)	04 i(8)	8086
ADD	AX/EAX	i(16/32)	05 i(16/32)	8086
ADD	mr(B)	i8	80 00+mr ?? i8	8086
ADD	mr(W/D)	is(8)	83 00+mr ?? is8	8086
ADD	mr(W/D)	i(16/32)	81 00+mr ?? i(16/32)	8086
ADD	mr(B)	r(B)	00 mrr ??	8086
ADD	mr(W/D)	r(W/D)	01 mrr ??	8086
ADD	r(B)	mr(B)	02 mrr ??	8086
ADD	r(W/D)	mr(W/D)	03 mrr ??	8086
AND	AL	i(8)	24 i(8)	8086
AND	AX/EAX	i(16/32)	25 i(16/32)	8086
AND	mr(B)	i(8)	80 20+mr ?? i(8)	8086
AND	mr(W/D)	i(16/32)	81 20+mr ?? i(16/32)	8086
AND	mr(B)	r(B)	20 mrr ??	8086
AND	mr(W/D)	r(W/D)	21 mrr ??	8086
AND	r(B)	mr(B)	22 mrr ??	8086
AND	r(W/D)	mr(W/D)	23 mrr ??	8086
ARPL	mr(W)	r(W)	63 mrr ??	286
BOUND	r(W/D)	m(D/Q)	62 mmr ??	186
BSF	r(W/D)	mr(W/D)	0F BC mrr ??	386
BSR	r(W/D)	mr(W/D)	0F BD mrr ??	386
BT	mr(W/D)	i(8)	0F BA 20+mr ?? i(8)	386
BT	mr(W/D)	r(W/D)	0F A3 mrr ??	386
BTC	mr(W/D)	i(8)	0F BA 38+mr ?? i(8)	386
BTC	mr(W/D)	r(W/D)	0F BB mrr ??	386
BTR	mr(W/D)	i(8)	0F BA 30+mr ??	386
BTR	mr(W/D)	r(W/D)	0F B3 mrr ??	386
BTS	mr(W/D)	i(8)	0F BA 28+mr ??	386
BTS	mr(W/D)	r(W/D)	0F AB mrr ??	386
CMP	AL	i(8)	3C i(8)	8086
CMP	AX/EAX	i(16/32)	3D i(16/32)	8086
CMP	mr(B)	is(8)	80 38+mr ?? is(8)	8086
CMP	mr(W/D)	i(8)	83 38+mr ?? i(8)	8086
CMP	mr(W/D)	i(16/32)	81 38+mr ?? i(16/32)	8086
CMP	mr(B)	r(B)	38 mrr ??	8086
CMP	mr(W/D)	r(W/D)	39 mrr ??	8086
CMP	r(B)	mr(B)	3A mrr ??	8086
CMP	r(W/D)	mr(W/D)	3B mrr ??	8086
CMPS	mr(B)	mr(B)	A6	8086
CMPS	mr(D)	mr(D)	## A7	386
CMPS	mr(W)	mr(W)	$$ A7	8086
CMPXCHG	mr(B)	r(B)	0F B0 mrr ??	486

アセンブラ用語集
イミディエイトオペランド　オペランドとしてレジスタ参照ではなく，直接16ビットまたは8ビットの値を指定すること

(S)mrr

レジスタ直接とレジスタ直接，レジスタ修飾あるいは直接アドレスの２つのオペランドをコード化して指定するバイトである．

(T)mrs

セグメントレジスタとレジスタ直接，レジスタ修飾あるいは直接アドレスの２つのオペランドをコード化して指定するバイトである．

最後のCPUの欄は，命令が正しく実行できる最下位のCPUを示している．以上がこの表の読み方であるが，この表を注意深く見ると次の特徴がわかるはずである．この特徴は，アセンブラを設計する上できわめて重要である．

(1)１つのニモニックの命令に対して，機械コードは必ずしも１つではない．オペランドまで調べないと機械コードは決定できない．この極端な例は表３のMOV命令である．

(2)各命令の機械コードの長さは一定していない．１つのニモニックの命令に対しても，オペランドを調べないと機械コードの長さは決定できない．これはオペランドを必要とする大半の命令にあてはまる．

(3)各命令の操作するオペランドのサイズは，オペランドを調べないと決定できない．アドレスサイズやイミーディエット（即値）のアドレッシングモードについても同様である．このことも(2)と同様，オペランドを必要とする多くの命令にあてはまる．

ニモニック	第1OPD	第2OPD	機械コード	CPU
CMPXCHG	mr(W/D)	r(W/D)	0F B1 mrr ??	486
ENTER	i(16)	i(8)	C8 i(16) i(8)	186
ESC	i(6)	mr(W/D)	D8+i(6) i(6)*8+mr ??	8086
IN	AL	DX	EC	8086
IN	AL	i(8)	E4 i(8)	8086
IN	AX/EAX	DX	ED	8086
IN	AX/EAX	i(8)	E5 i(8)	8086
INS	mr(B)	DX	6C	186
INS	mr(W)	DX	$$ 6D	386
INS	mr(D)	DX	## 6D	186
LAR	r(W/D)	mr(W/D)	0F 02 mrr ??	286
LDS	r(W/D)	m(D/F)	C5 mmr ??	8086
LEA	r(W/D)	m(W/D)	8D mmr ??	8086
LES	r(W/D)	m(D/F)	C4 mmr ??	8086
LFS	r(W/D)	m(D/F)	0F B4 mmr ??	386
LGS	r(W/D)	m(D/F)	0F B5 mmr ??	386
LSL	r(W/D)	mr(W/D)	0F 03 mrr ??	286
LSS	r(W/D)	m(D/F)	0F B2 mmr ??	386
MOV	AL	l16/32	A0 l16/32	8086
MOV	AX/EAX	l16/32	A1 l16/32	8086
MOV	mr(B)	i(8)	C6 00+mr ?? i(8)	8086
MOV	mr(W/D)	i(16/32)	C7 00+mr ?? i(16/32)	8086
MOV	mr(B)	r(B)	88 mrr ??	8086
MOV	mr(W/D)	r(W/D)	89 mrr ??	8086
MOV	mr(W/D)	s3	8C mrs ??	8086
MOV	l16/32	AL	A2 l16/32	8086
MOV	l16/32	AX/EAX	A3 l16/32	8086
MOV	r(B)	mr(B)	8A mrr ??	8086
MOV	r(B)	i(8)	B0+r i(8)	8086
MOV	r(W/D)	mr(W/D)	8B mrr ??	8086
MOV	r(W/D)	i(16/32)	B8+r i(16/32)	8086
MOV	s3	mr(W/D)	8E mrs ??	8086
MOV	CRn	r(D)	0F 22 C0+n*8+r	386
MOV	r(D)	CRn	0F 20 C0+n*8+r	386
MOV	DRn	r(D)	0F 23 C0+n*8+r	386
MOV	r(D)	DRn	0F 21 C0+n*8+r	386
MOV	TRn	r(D)	0F 26 C0+n*8+r	386
MOV	r(D)	TRn	0F 24 C0+n*8+r	386
MOVS	mr(B)	mr(B)	A4	8086
MOVS	mr(D)	mr(D)	## A5	386
MOVS	mr(W)	mr(W)	$$ A5	8086
MOVSX	r(W)	mr(B)	$$ 0F BE mrr ??	386
MOVSX	r(D)	mr(W)	## 0F BF mrr ??	386
MOVZX	r(W)	mr(B)	$$ 0F B6 mrr ??	386
MOVZX	r(D)	mr(W)	## 0F B7 mrr ??	386
OR	AL	i(8)	0C i(8)	8086
OR	AX/EAX	i(16/32)	0D i(16/32)	8086
OR	mr(B)	i(8)	80 08+mr ?? i(8)	8086
OR	mr(W/D)	i(16/32)	81 08+mr ?? i(16/32)	8086
OR	mr(B)	r(B)	08 mrr ??	8086
OR	mr(W/D)	r(W/D)	09 mrr ??	8086
OR	r(B)	mr(B)	0A mrr ??	8086
OR	r(W/D)	mr(W/D)	0B mrr ??	8086

アセンブラ用語集
インクルード　include疑似命令などでヘッダやソースファイルをプログラムに挿入すること。

ニモニック	第1OPD	第2OPD	機械コード	CPU
OUT	DX	AL	EE	8086
OUT	i(8)	AL	E6 i(8)	8086
OUT	DX	AX/EAX	EF	8086
OUT	i(8)	AX/EAX	E7 i(8)	8086
OUTS	DX	mr(B)	6E	186
OUTS	DX	mr(D)	## 6F	386
OUTS	DX	mr(W)	$$ 6F	186
RCL	mr(B)	1	D0 10+mr ??	8086
RCL	mr(B)	i(8)	C0 10+mr ?? i(8)	186
RCL	mr(B)	CL	D2 10+mr ??	8086
RCL	mr(W/D)	1	D1 10+mr ??	8086
RCL	mr(W/D)	i(8)	C1 10+mr ?? i(8)	186
RCL	mr(W/D)	CL	D3 10+mr ??	8086
RCR	mr(B)	1	D0 18+mr ??	8086
RCR	mr(B)	i(8)	C0 18+mr ?? i(8)	186
RCR	mr(B)	CL	D2 18+mr ??	8086
RCR	mr(W/D)	1	D1 18+mr ??	8086
RCR	mr(W/D)	i(8)	C1 18+mr ?? i(8)	186
RCR	mr(W/D)	CL	D3 18+mr ??	8086
ROL	mr(B)	1	D0 00+mr ??	8086
ROL	mr(B)	i(8)	C0 00+mr ?? i(8)	186
ROL	mr(B)	CL	D2 00+mr ??	8086
ROL	mr(W/D)	1	D1 00+mr ??	8086
ROL	mr(W/D)	i(8)	C1 00+mr ?? i(8)	186
ROL	mr(W/D)	CL	D3 00+mr ??	8086
ROR	mr(B)	1	D0 08+mr ??	8086
ROR	mr(B)	i(8)	C0 08+mr ?? i(8)	186
ROR	mr(B)	CL	D2 08+mr ??	8086
ROR	mr(W/D)	1	D1 08+mr ??	8086
ROR	mr(W/D)	i(8)	C1 08+mr ?? i(8)	186
ROR	mr(W/D)	CL	D3 08+mr ??	8086
SAL	mr(B)	1	D0 20+mr ??	8086
SAL	mr(B)	i(8)	C0 20+mr ?? i(8)	186
SAL	mr(B)	CL	D2 20+mr ??	8086
SAL	mr(W/D)	1	D1 20+mr ??	8086
SAL	mr(W/D)	i(8)	C1 20+mr ?? i(8)	186
SAL	mr(W/D)	CL	D3 20+mr ??	8086
SAR	mr(B)	1	D0 38+mr ??	8086
SAR	mr(B)	i(8)	C0 38+mr ?? i(8)	186
SAR	mr(B)	CL	D2 38+mr ??	8086
SAR	mr(W/D)	1	D1 38+mr ??	8086
SAR	mr(W/D)	i(8)	C1 38+mr ?? i(8)	186
SAR	mr(W/D)	CL	D3 38+mr ??	8086
SBB	AL	i(8)	1C i(8)	8086
SBB	AX/EAX	i(16/32)	1D i(16/32)	8086
SBB	mr(B)	i(8)	80 18+mr ?? i(8)	8086
SBB	mr(W/D)	is(8)	83 18+mr ?? is(8)	8086
SBB	mr(W/D)	i(16/32)	81 18+mr ?? i(16/32)	8086
SBB	mr(B)	r(B)	18 mrr ??	8086
SBB	mr(W/D)	r(W/D)	19 mrr ??	8086
SBB	r(B)	mr(B)	1A mrr ??	8086
SBB	r(W/D)	mr(W/D)	1B mrr ??	8086
SHL	mr(B)	1	D0 20+mr ??	8086

(4) 1つのニモニックの命令に対して，CPUによって使用できるオペランドとそうでないオペランドが存在する．たとえば，表2のPUSH命令やRCLなどのシフト系の命令，表3のMOV命令などである．

(5)異なる命令の間には，まったく同様のオペランドをとる命令が存在するので，機械コードを生成する際にグループ化して共通の処理をすることができる．たとえば，表1のJA，JAEなどをはじめとする条件付きジャンプ命令，あるいは表2のADC，ADD，CMP，SBBおよびSUB命令などがこれに該当する．

(6)プリフィックス命令やセグメントオーバーライド命令のように通常のアセンブラにはあまり見られない処理が必要な命令がある．プリフィックス命令には，ASIZE，LOCK，OSIZE，REP，REPE，REPNE，REPNZおよびREPZがある．とくに，ASIZEとOSIZEはアセンブラが自動的に処理する必要がある．セグメントオーバーライド命令には，CS:，DS:，ES:，FS:，GS:，SS:の6つがあるが，これらの命令はオペランドの中に記述される命令であり，特別な処理が必要となる．

(7)同じニモニックの命令の中で，特定のオペランドに対し2つ以上の機械コードが存在し，どちらの機械コードを使用しても構わない命令が存在する．とくに，表2のINC，DEC，PUSHやPOP命令など一部の命令では，同じオペランドに対して機械コードの長さが異なる．このような命令では，アセンブラはコードサイズを節約するため，自動的に短いコードを選択するのが通例である．

アセンブラ用語集
インストラクションポインタ　CPUが次に実行するべき命令が入っているアドレスが格納されている．

ニモニック	第1OPD	第2OPD	機械コード	CPU
SHL	mr(B)	i(8)	C0 20+mr ?? i(8)	186
SHL	mr(B)	CL	D2 20+mr ??	8086
SHL	mr(W/D)	1	D1 20+mr ??	8086
SHL	mr(W/D)	i(8)	C1 20+mr ?? i(8)	186
SHL	mr(W/D)	CL	D3 20+mr ??	8086
SHR	mr(B)	1	D0 28+mr ??	8086
SHR	mr(B)	i(8)	C0 28+mr ?? i(8)	186
SHR	mr(B)	CL	D2 28+mr ??	8086
SHR	mr(W/D)	1	D1 28+mr ??	8086
SHR	mr(W/D)	i(8)	C1 28+mr ?? i(8)	186
SHR	mr(W/D)	CL	D3 28+mr ??	8086
SUB	AL	i(8)	2C i(8)	8086
SUB	AX/EAX	i(16/32)	2D i(16/32)	8086
SUB	mr(B)	i(8)	80 28+mr ?? i(8)	8086
SUB	mr(W/D)	is(8)	83 28+mr ?? is(8)	8086
SUB	mr(W/D)	i(16/32)	81 28+mr ?? i(16/32)	8086
SUB	mr(B)	r(B)	28 mrr ??	8086
SUB	mr(W/D)	r(W/D)	29 mrr ??	8086
SUB	r(B)	mr(B)	2A mrr ??	8086
SUB	r(W/D)	mr(W/D)	2B mrr ??	8086
TEST	AL	i(8)	A8 i(8)	8086
TEST	AX/EAX	i(16/32)	A9 i(16/32)	8086
TEST	mr(B)	i(8)	F6 00+mr ?? i(8)	8086
TEST	mr(W/D)	i(16/32)	F7 00+mr ?? i(16/32)	8086
TEST	mr(B)	r(B)	84 mrr ??	8086
TEST	mr(W/D)	r(W/D)	85 mrr ??	8086
XADD	mr(B)	r(B)	0F C0 mrr ??	486
XADD	mr(W/D)	r(W/D)	0F C1 mrr ??	486
XCHG	AX/EAX	r(W/D)	90+r	8086
XCHG	r(B)	mr(B)	86 mrr ??	8086
XCHG	r(W/D)	mr(W/D)	87 mrr ??	8086
XOR	AL	i(8)	34 i(8)	8086
XOR	AX/EAX	i(16/32)	35 i(16/32)	8086
XOR	mr(B)	i(8)	80 30+mr ?? i(8)	8086
XOR	mr(W/D)	i(16/32)	81 30+mr ?? i(16/32)	8086
XOR	mr(B)	r(B)	30 mrr ??	8086
XOR	mr(W/D)	r(W/D)	31 mrr ??	8086
XOR	r(B)	mr(B)	32 mrr ??	8086
XOR	r(W/D)	mr(W/D)	33 mrr ??	8086

注：記号の説明は「1-3」を参照

(8)オペランドのサイズやアドレッシングモードを自動的に判別するためには，シンボルに特別の情報が必要である．

「1-1」でシンボルがラベル，バリアブルおよびナンバーに分類されることを説明したが，この分類はこのためである．さらに，ラベルとバリアブルにはオペランドのサイズを決定する情報が必要である．ラベルは，命令の実行順序を変更するCALLやJMPなどの命令に使用される．これに対して，バリアブルはメモリのデータを操作する命令のオペランドに使用される．ナンバーは，命令が通常のメモリに格納されているデータではなく，命令自身に含まれるデータをオペランドとするイミーディエットアドレッシングモードであることを識別するのに必要である．

1-4 機械コードの構成

表1～表4では，いきなりすべての命令と機械コードの対応を示したので多少わかりにくかったかもしれない．そこで，これらの表の機械コードを整

表4　オペランドを3つ必要とするプロセッサ命令

ニモニック	第1OPD	第2OPD	第3OPD	機械コード	CPU
IMUL	mr(B)			F6 28+mr ??	8086
IMUL	mr(W/D)			F7 28+mr ??	8086
IMUL	r(W/D)	mr(W/D)	i(8)	6B mrr ?? i(8)	186
IMUL	r(W/D)	mr(W/D)	i(16/32)	69 mrr ?? i(16/32)	186
IMUL	r(W/D)	mr(W/D)		0F AF mrr ??	386
SHLD	mr(W/D)	r(W/D)	i(8)	0F A4 mrr ?? i(8)	386
SHLD	mr(W/D)	r(W/D)	CL	0F A5 mrr ??	386
SHRD	mr(W/D)	r(W/D)	i(8)	0F AC mrr ?? i(8)	386
SHRD	mr(W/D)	r(W/D)	CL	0F AD mrr ??	386

注：記号の説明は「1-3」を参照

アセンブラ用語集
インデックスレジスタ　ストリング転送命令などで，メモリ中の配列やテーブルをアクセスしたときに参照されるレジスタ．

理して8086系のCPUの機械コードの構成を示すと，図1aと図1bのように2つに分類して表すことができる．

図1a　286以下のCPUと386以上のCPUの16ビットサイズのときの命令コード

tttttttt	[tttttttt]	[mdregmem]	[disp8/16]	[immd8/16]

図1b　386以上のCPUの32ビットサイズのときの命令コード

tttttttt	[tttttttt]	[mdregmem]	[scixrbas]	[disp8/32]	[immd8/32]

1-4-1 16ビットサイズと32ビットサイズ

図1aと図1bの2つの図を示してあるのは，386以上のCPUではアドレスサイズとオペランドサイズが16ビットサイズと32ビットサイズの2つの動作モードを持っているからである．図1aは，286以下のCPUと386以上のCPUの16ビットサイズの動作モードでの機械コードの構成であり，図1bは，386以上のCPUの32ビットサイズの動作モードのときの機械コードの構成である．

386以上の16ビットサイズの動作モードは，286以下のCPUと互換性のある動作モードで，386以上のCPUでもリアルモードと286互換のプロテクトモードおよび仮想86モードはこのモードで動作する．もちろん，MS-DOSから起動されたプログラムもこのモードで動作する．しかし，386以上のCPUでは16ビットモードで動作中も一時的にアドレスサイズとオペランドサイズを，個別に32ビットサイズに切り替えることができるようになっている．この切り替えを行うのが，「1-3」でお話したASIZEとOSIZEの2つのプリフィックス命令である．サイズの切り替えは，プリフィックス命令の付いた1命令だけに限って有効である．また，2つのプリフィックスを同時に使用することも可能である．

サイズを切り替えることのできる命令は，表1～表4のオペランド欄で確認できる．()の中に/で区切って2つのサイズが示してある命令は，オペランドのサイズを切り替えることができる．これに該当する命令は，16ビットサイズの動作モードのとき1ワードサイズ以上のオペランドを持つ命令である．ただし，ENTER命令やRET命令などは例外である．d16/32あるいはi16/32のように書いてある命令と，mやmrをオペランドに持つ命令はアドレスサイズの切り替えが可能である．

32ビットサイズの動作モードでは，アドレスサイズもオペランドサイズも32ビットである．この動作モードは，386以上のCPUに固有なプロテクトモードでのみ利用可能である．この動作モードのときASIZEとOSIZEのプリフィックス命令を使用すると，16ビットサイズ動作モードのときとは逆に，アドレスサイズやオペランドサイズを一時的に16ビットサイズに切り替えることができる．

1-4-2 命令コード(tttttttt)

図1aと図1bに示したttttttttは，命令コードである．命令コードは表1～表4の機械コードの欄に数字と大文字の16進数で表示されている部分で，命令によって1～2バイトの長さである．命令コードは，ニモニックとオペランドのタイプが決まれば，決定することができる．1部の命令，たとえばDIVやMUL命令などでは次に述べるmdregmemバイトのregの3ビットにも命令コードのあるものがある．もし，プリフィックス命令やセグメントオーバーライド命令が必要なときは，それらの命令はttttttttの直前にこなければならない．

1-4-3 mdregmemバイトとscixrbsrバイト

多くの命令では，オペランドのアドレッシングモードを選ぶことができる．そのような命令には，このmdregmemバイトが命令コードの直後に来る．また，一部のアドレッシングモードが固定の命令では，このmdregmemバイトは省略される．mdregmemバイトは，1文字を1ビットに対応させたコードで，2ビットのmdと3ビットのregとmemの3つの部分に分解することができる．

2つ以上のオペランドが必要な命令では，一方のオペランドはレジスタに固定されるので，regはレジスタに固定されているオペランドのレジスタコ

アセンブラ用語集
インデント　行の先頭にタブや空白を入れること．プログラミングスタイルのひとつでソースファイルでプログラムの流れを判断しやすくする．

ードが入る．1つのオペランドだけを必要とする命令では，regの部分は命令コードの1部であったり，000B固定であったりする．regの部分は，16ビットと32ビットサイズのどちらのモードにも共通である．この部分に使用されるレジスタとコードの対応表は表5である．mdとmemの部分は，もう一方のアドレッシングモードを示す5ビットのコードである．この5ビットのコードは，16ビットサイズの動作モードと32ビットサイズの動作モードで異なっている．2つのサイズのアドレッシングモードとmdregmemバイトのコードとの対応は，それぞれ表6と表7の通りである．

表6と表7においてmd=11Bの場合は，16ビットサイズのときも32ビットサイズのときもレジスタ直接となる．表6のmd=00Bでmem=110Bの場合は，16ビットサイズにおける直接アドレスである．また，表7のmd=00Bでmem=101Bの場合が，32ビットサイズにおける直接アドレスである．これら以外の場合は，レジスタ修飾となる．

scixrbsrバイト（s-i-bバイト）は，アドレスサイズが32ビットの特別なアドレッシングモードのときにのみ必要となるバイトである．このバイトがあるときは，必ず先行するmdregmemバイトが必要である．またこの場合，先行するmdregmemバイトのmdが11B以外でmemが100Bである必要がある．scixrbsrもmdregmemと同様，1文字1ビットに対応させたコードで，2ビットのscと3ビットのixrとbsrの3つの部分に分かれる．このバイトが使用されるのは，スケールドインデックスモードとESPをベースレジスタに使用する場合である．表8aと表8bにアドレッシングモードとscixrbsrバイトのコードとの対応を示す．また表9はscのコード（表8aと表8bの＊sに対応）を示す．

表5　オペランドに指定できるレジスタのコード

コード	BYTE REGISTER	WORD REGISTER	SEGMENT REGISTER	DWORD REGISTER	CONTROL REGISTER	DEBUG REGISTER	TEST REGISTER
000	AL	AX	ES	EAX＊	CR0＊	DR0＊	—
001	CL	CX	CS	ECX＊	—	DR1＊	—
010	DL	DX	SS	EDX＊	CR2＊	DR2＊	—
011	BL	BX	DS	EBX＊	CR3＊	DR3＊	TR3#
100	AH	SP	FS＊	ESP＊	—	—	TR4#
101	CH	BP	GS＊	EBP＊	—	—	TR5#
110	DH	SI	—	ESI＊	—	DR6＊	TR6＊
111	BH	DI	—	EDI＊	—	DR7＊	TR7＊

注：＊は386以上，#は486のみで使用可能

表6　サイズ16ビットのmdregmemバイトのコード

md	mem 000	001	010	011	100	101	110	111
00	[BX+SI]	[BX+DI]	[BP+SI]	[BP+DI]	[SI]	[DI]	d16	[BX]
01	d8[BX+SI]	d8[BX+DI]	d8[BP+SI]	d8[BP+DI]	d8[SI]	d8[DI]	d8[BP]	d8[BX]
10	d16[BX+SI]	d16[BX+DI]	d16[BP+SI]	d16[BP+DI]	d16[SI]	d16[DI]	d16[BP]	d16[BX]
11	AL/AX	CL/CX	DL/DX	BL/BX	AH/SP	CH/BP	DH/SI	BH/DI

表7　32ビットサイズのmdregmemバイトのコード（scixrbsrバイトのないとき）

md	mem 000	001	010	011	100	101	110	111
00	[EAX]	[ECX]	[EDX]	[EBX]	scixrbsr	d32	[ESI]	[EDI]
01	d8[EAX]	d8[ECX]	d8[EDX]	d8[EBX]	scixrbsr	d8[EBP]	d8[ESI]	d8[EDI]
10	d32[EAX]	d32[ECX]	d32[EDX]	d32[EBX]	scixrbsr	d32[EBP]	d32[ESI]	d32[EDI]
11	AL/EAX	CL/ECX	DL/EDX	BL/EBX	AH/ESP	CH/EBP	DH/ESI	BH/EDI

アセンブラ用語集
インラインアセンブラ　C言語などほかの言語でソースファイル中のアセンブラソースをアセンブルする機能

表 8a　32 ビットサイズの mdregmem と scixrbsr バイトのコード(1)

md	bsr	ixr			
		000	001	010	011
00	000	[EAX+EAX * s]	[EAX+ECX * s]	[EAX+EDX * s]	[EAX+EBX * s]
00	001	[ECX+EAX * s]	[ECX+ECX * s]	[ECX+EDX * s]	[ECX+EBX * s]
00	010	[EDX+EAX * s]	[EDX+ECX * s]	[EDX+EDX * s]	[EDX+EBX * s]
00	011	[EBX+EAX * s]	[EBX+ECX * s]	[EBX+EDX * s]	[EBX+EBX * s]
00	100	[ESP+EAX * s]	[ESP+ECX * s]	[ESP+EDX * s]	[ESP+EBX * s]
00	101	[EBP+EAX * s]	[EBP+ECX * s]	[EBP+EDX * s]	[EBP+EBX * s]
00	110	[ESI+EAX * s]	[ESI+ECX * s]	[ESI+EDX * s]	[ESI+EBX * s]
00	111	[EDI+EAX * s]	[EDI+ECX * s]	[EDI+EDX * s]	[EDI+EBX * s]
01	000	d8[EAX+EAX * s]	d8[EAX+ECX * s]	d8[EAX+EDX * s]	d8[EAX+EBX * s]
01	001	d8[ECX+EAX * s]	d8[ECX+ECX * s]	d8[ECX+EDX * s]	d8[ECX+EBX * s]
01	010	d8[EDX+EAX * s]	d8[EDX+ECX * s]	d8[EDX+EDX * s]	d8[EDX+EBX * s]
01	011	d8[EBX+EAX * s]	d8[EBX+ECX * s]	d8[EBX+EDX * s]	d8[EBX+EBX * s]
01	100	d8[ESP+EAX * s]	d8[ESP+ECX * s]	d8[ESP+EDX * s]	d8[ESP+EBX * s]
01	101	d8[EBP+EAX * s]	d8[EBP+ECX * s]	d8[EBP+EDX * s]	d8[EBP+EBX * s]
01	110	d8[ESI+EAX * s]	d8[ESI+ECX * s]	d8[ESI+EDX * s]	d8[ESI+EBX * s]
01	111	d8[EDI+EAX * s]	d8[EDI+ECX * s]	d8[EDI+EDX * s]	d8[EDI+EBX * s]
10	000	d32[EAX+EAX * s]	d32[EAX+ECX * s]	d32[EAX+EDX * s]	d32[EAX+EBX * s]
10	001	d32[ECX+EAX * s]	d32[ECX+ECX * s]	d32[ECX+EDX * s]	d32[ECX+EBX * s]
10	010	d32[EDX+EAX * s]	d32[EDX+ECX * s]	d32[EDX+EDX * s]	d32[EDX+EBX * s]
10	011	d32[EBX+EAX * s]	d32[EBX+ECX * s]	d32[EBX+EDX * s]	d32[EBX+EBX * s]
10	100	d32[ESP+EAX * s]	d32[ESP+ECX * s]	d32[ESP+EDX * s]	d32[ESP+EBX * s]
10	101	d32[EBP+EAX * s]	d32[EBP+ECX * s]	d32[EBP+EDX * s]	d32[EBP+EBX * s]
10	110	d32[ESI+EAX * s]	d32[ESI+ECX * s]	d32[ESI+EDX * s]	d32[ESI+EBX * s]
10	111	d32[EDI+EAX * s]	d32[EDI+ECX * s]	d32[EDI+EDX * s]	d32[EDI+EBX * s]

表 8b　32 ビットサイズの mdregmem と scixrbsr バイトのコード(2)

md	bsr	ixr			
		100	101	110	111
00	000	[EAX]	[EAX+EBP * s]	[EAX+ESI * s]	[EAX+EDI * s]
00	001	[ECX]	[ECX+EBP * s]	[ECX+ESI * s]	[ECX+EDI * s]
00	010	[EDX]	[EDX+EBP * s]	[EDX+ESI * s]	[EDX+EDI * s]
00	011	[EBX]	[EBX+EBP * s]	[EBX+ESI * s]	[EBX+EDI * s]
00	100	[ESP]	[ESP+EBP * s]	[ESP+ESI * s]	[ESP+EDI * s]
00	101	[EBP]	[EBP+EBP * s]	[EBP+ESI * s]	[EBP+EDI * s]
00	110	[ESI]	[ESI+EBP * s]	[ESI+ESI * s]	[ESI+EDI * s]
00	111	[EDI]	[EDI+EBP * s]	[EDI+ESI * s]	[EDI+EDI * s]
01	000	d8[EAX]	d8[EAX+EBP * s]	d8[EAX+ESI * s]	d8[EAX+EDI * s]
01	001	d8[ECX]	d8[ECX+EBP * s]	d8[ECX+ESI * s]	d8[ECX+EDI * s]
01	010	d8[EDX]	d8[EDX+EBP * s]	d8[EDX+ESI * s]	d8[EDX+EDI * s]
01	011	d8[EBX]	d8[EBX+EBP * s]	d8[EBX+ESI * s]	d8[EBX+EDI * s]
01	100	d8[ESP]	d8[ESP+EBP * s]	d8[ESP+ESI * s]	d8[ESP+EDI * s]
01	101	d8[EBP]	d8[EBP+EBP * s]	d8[EBP+ESI * s]	d8[EBP+EDI * s]
01	110	d8[ESI]	d8[ESI+EBP * s]	d8[ESI+ESI * s]	d8[ESI+EDI * s]
01	111	d8[EDI]	d8[EDI+EBP * s]	d8[EDI+ESI * s]	d8[EDI+EDI * s]
10	000	d32[EAX]	d32[EAX+EBP * s]	d32[EAX+ESI * s]	d32[EAX+EDI * s]
10	001	d32[ECX]	d32[ECX+EBP * s]	d32[ECX+ESI * s]	d32[ECX+EDI * s]
10	010	d32[EDX]	d32[EDX+EBP * s]	d32[EDX+ESI * s]	d32[EDX+EDI * s]
10	011	d32[EBX]	d32[EBX+EBP * s]	d32[EBX+ESI * s]	d32[EBX+EDI * s]

md	bsr	ixr 100	01	10	11
10	100	d32[ESP]	d32[ESP+EBP ＊ s]	d32[ESP+ESI ＊ s]	d32[ESP+EDI ＊ s]
10	101	d32[EBP]	d32[EBP+EBP ＊ s]	d32[EBP+ESI ＊ s]	d32[EBP+EDI ＊ s]
10	110	d32[ESI]	d32[ESI+EBP ＊ s]	d32[ESI+ESI ＊ s]	d32[ESI+EDI ＊ s]
10	111	d32[EDI]	d32[EDI+EBP ＊ s]	d32[EDI+ESI ＊ s]	d32[EDI+EDI ＊ s]

表9　スケールのコード

sc	スケール
00	＊1
01	＊2
10	＊4
11	＊8

scは，スケールドインデックス以外のときは00Bでなければならない．

1-4-4 変位(disp8/16 disp8/32)

レジスタ修飾の変位，直接アドレスあるいはIP相対のオフセットがこの位置にくる．アドレッシングモードが固定の命令では，先行するmdregmemバイトやscixrbsrバイトはないことがある．32ビットのアドレスサイズでは，8ビットと32ビットが，16ビットのアドレスサイズでは，8ビットと16ビットが変位の長さとなる．どちらのアドレスサイズでも，変位のないレジスタ修飾モードのときはこの変位は存在しない．

1-4-5 イミーディエットデータ(immd8/16 immd8/32)

イミーディエットのオペランドがあるときは，機械コードの最後にイミーディエットデータがおかれる．ただしESC命令は例外である．32ビットのアドレスサイズでは，8ビットと32ビットが，16ビットのアドレスサイズでは，8ビットと16ビットがイミーディエットデータの長さとなる．

1-4-6 機械コードの例

手作業で機械コードをコード化することをハンドアセンブルと呼ぶ．ハンドアセンブルが間違いなくできるようになると，アセンブラの作業のかなりの部分が理解できる．以下にハンドアセンブルの例をいくつかあげてみよう(例2，3，4，5)．

例2

```
PUSH    -1                  = 6A FF

ttttttt = 6A                ; PUSH(-1はis(8)の範囲に入る)
immd8   = FF                ; -1の2の補数
```

例3

```
MOV     CX,SI               = 89 F1, 8B CE

ttttttt = 89                ; MOV(8Bでもよい)
md      = 11                ; レジスタ直接(CX), 8Bのときは(SI)
reg     = 110               ; レジスタ直接(SI), 001(CX)
mem     = 001               ; レジスタ直接(CX), 110(SI)
```

例4

```
LSS     SI,2000H[BX+DI]     = 0FB2 B1 0020

ttttttt = 0FB2              ; LSS
md      = 10                ; d16[bsr+ixr]
reg     = 110               ; レジスタ直接(SI)
mem     = 001               ; d16[BX+DI]
disp16  = 0020              ; 下位が先で,上位が後
```

例5

```
ADC     EAX,[ESP]           = 13 04 24

ttttttt = 13                ; ADC
md      = 00                ; [bsr]
reg     = 000               ; EAX
mem     = 100               ; scixrbsr必要
sc      = 00                ; 00以外指定できない
ixr     = 100               ; [bsr]
bsr     = 100               ; [ESP]
```


アセンブラ用語集
オーバーヘッド　演算などが実行されない時間のこと．サブルーチンはマクロ参照よりサイズは小さくなるがオーバーヘッドが大きい．

特集 アセンブラの作り方

2 アセンブラ処理の概要と簡易アセンブラの処理方式

アセンブラで一般によく使用されるアルゴリズムは，２パス方式と呼ばれるものである．絶対形式のアセンブラでも，リロケータブルなアセンブラでも，マクロアセンブラでもこの方式のものが多い．パスはフェーズとも呼ばれ，ソースを始めから終わりまで走査して処理を行うことである．

アセンブラのパス1では，ソースプログラムを走査して，おもにシンボルテーブルが作成される．次の例６のように，シンボルの参照がシンボルが定義される以前に行われると，アセンブラは機械コードを生成することができない．

そこで，アセンブラのパス１では，ソースプログラムの１行ごとの命令の語数を数えながら命令の先頭アドレスを決定する．この数えた語数を累積して，命令の先頭アドレスを保持するためのカウンタを位置割当カウンタ(LAC)と呼ぶ．命令の前にシンボルが付けられて定義されると，LACの値をシンボルの値としてシンボルテーブルに登録する．8086系のアセンブラでは，シンボルの付けられた命令や強制的に値を与えられた状況に応じて，シンボルがラベルなのかバリアブルなのかナンバーなのかという属性もシンボルテーブルに値とともに登録する．バリアブルのサイズについても同様のことが行われる．パス１によってシンボルテーブルが作成されると，アセンブラは機械コード生成の準備が完了する．

パス２では，再びソースプログラムを走査して，オブジェクトプログラムとアセンブリリストを作成する．機械コードの生成は，前方参照のシンボルが存在してもパス１で作成したシンボルテーブルを参照することで解決する．以上が２パス式アセンブラのアルゴリズムで，この様子を図２に示す．

さて，賢明な読者諸氏にはおわかりになると思うが，２パス式のアセンブラを8086系CPUのアセンブリ言語に適用しようとすると，問題が発生するのである．実は，8086系CPUのアセンブリ言語では，オペランドが決まらないと機械コードが決定できないばかりか，命令のバイト数も決められないはずである．パス１でアセンブラが前方参照のシンボルに出会ったとき，どのようにして語数を決定するのであろうか．この解決策は，もし前方参照のシンボルに出会ったならデフォルトの仮定を行い，語数を決めることである．ただし，デフォルトの仮定をうまく行わないと，パス１とパス２でシンボルの値が異なるというフェーズエラーが発生することになる．

もう１つの解決策として考えられるのが，パスを１つ増やして３パス方式にすることである．筆者の作成した簡易アセンブラは，この３パス方式を採

例６

```
        MOV     AX,RETCODE      ; RETCODEの前方参照
        MOV     AH,4CH
        INT     MSDOS
............................
............................
RETCODE DW      0               ; RETCODEの定義
```

アセンブラ用語集
オフセット　ある基準（セグメント）からの相対的な位置のこと．メモリの位置を示すのに使われる．

用している．パスを1つ追加して3パス式にすると，2パス式に比べて，速度が1.5倍程度遅くなるが，パス1にあまり神経を使わなくて済むので設計が楽になるからである．ただし，これは個人使用の簡易アセンブラだからできることで，アセンブラにとって速度が遅くなることは好ましいことではない．

3パス方式のアセンブラのパス1での作業は，2パス方式のアセンブラとほぼ同じである．前方参照のシンボルに出会ったときは，命令のニモニックや別のオペランドから判明している情報を使用して，できる限りバイト長の大きい機械コードを仮定して，命令のバイト数を決定する．これを元にパス2用のシンボルテーブルを作成する．

パス2では，パス1で作成されたシンボルテーブルを使用して，今度は可能な限り短いバイト長の機械コードを使用して命令のバイト数を決定する．ソースプログラムの書き方にもよるが，パス2ではほとんどの命令について正しくバイト数を決定できるはずである．このバイト数を元にシンボルテーブルにパス2でのシンボルの値を加える．パス2での命令のバイト長を保存するために，シンボルテーブルにはパス1の値を残しておく．

パス3では，シンボルテーブルのパス1とパス2の値を使用してオブジェクトプログラム，アセンブリリストを作成する．パス2とパス3のフェーズエラーが発生しないようパス1で作成した値を使用して命令のバイト数を決定し，実際の値はパス2の値を使用して機械コードを生成する．このようにすると冗長な機械コードを生成することもあるが，通常のソースコードでは希である．以上の処理をまとめると，図3の通りである．

処理方式を説明したついでに，簡易アセンブラの仕様を簡単に紹介しておこう．

(1)アセンブラの形式と処理方式

絶対形式の3パス式アセンブラ（マクロ機能なし）．

(2)アセンブル可能なプロセッサ命令

8086から486CPUの全命令．ただし，32ビットの直接アドレスアドレッシングモードは使用できない．また，32ビットサイズ動作モード用のコードは出力できない．

(3)プログラミング可能なメモリモデル

タイニーモデルのみ，したがって複数のセグメントにまたがるアセンブル

図2　2パス式アセンブル処理

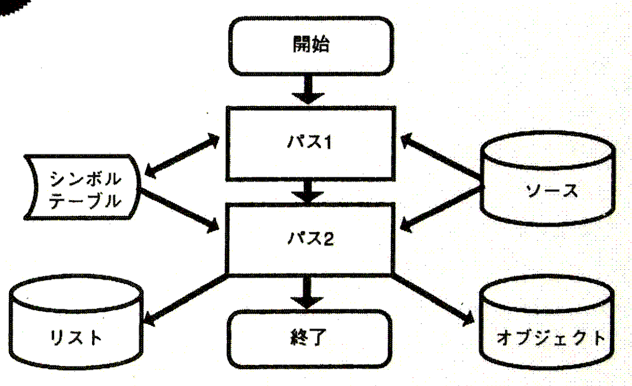
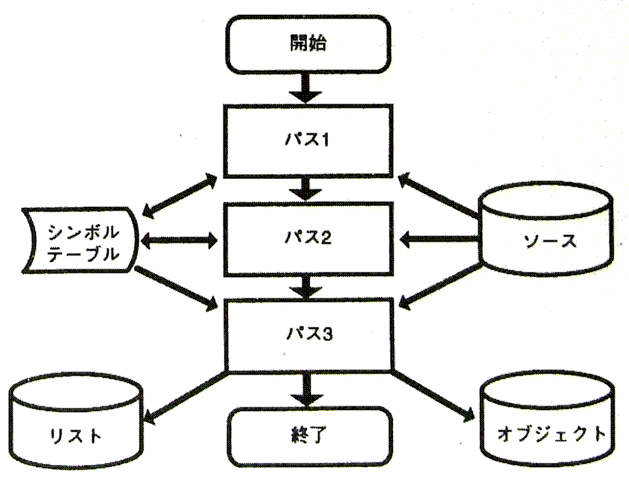


図3　簡易アセンブラの処理方式



アセンブラ用語集
オフセットアドレス　実アドレスの下位16ビットの値で，4ビットシフトさせたセグメントアドレスとの和が実アドレスとなる．

はサポートしない．

(4)ラベルの最大文字数

31文字．

(5)使用できるディレクティブ

表10の通り．

市販されているマクロアセンブラに比べればかなり見劣りする仕様であるが，チョイ書きのツールやデバイスドライバの作成などには十分である．

表10　製作したアセンブラのディレクティブ

ニモニック	オペランド	機械コード
.186	なし	なし
.286	なし	なし
.386	なし	なし
.486	なし	なし
.8086	なし	なし
DB DD DW	任意の数のオペランド (ただし,?と DUP 演算子はサポートしていない)DB は NUM を表す式,DD と DW は NUM,VAR,LBL を表す式	オペランドのデータをそのままオブジェクトとする
ELSE	なし	なし
END	なし	なし
ENDIF	なし	なし
EQU	NUM,VAR,LBL を表す式	なし
IF	NUM を表す式	なし
ORG	NUM,VAR,LBL を表す式	なし

特集 アセンブラの作り方

3 アセンブラの制御

さて，いよいよ簡易アセンブラのプログラムの紹介に入る．簡易アセンブラの記述言語はMS-C6.0である．プログラムのソースファイルは，AS86.H，AS86.C，FIO86.C，LBL86.C，EXP86.C，OP86.C，PX86.Cの7つに分かれている．以下の説明で不明の点は，ソースリストを参照していただければ幸いである．

A プログラム1 (AS86.H)

プログラム1（AS86.H）には，簡易アセンブラの記述で使用するグローバルな定数，構造体，変数および関数が宣言・定義されている．ここに記述されている内容は，プログラムを読む上で重要なものなので，少し長くなるが以下に全部を簡単に説明しておく．

(1) VERSION，LEVEL

簡易アセンブラ自身の主バージョンと，副バージョンの番号であり，起動時のメッセージとアセンブリリストに表示される．

アセンブラ用語集
オペランド　ほとんどの命令は，ふたつデータ間での処理を行う．この命令の実行対象となるものをオペランドという．

(2) IFMAX

IF～ELSE～ENDIFブロックのネスト可能な最大レベルを設定する．

(3) SRCMAX

ソースプログラム1行の終端記号(CRLF)を除いた最大文字数である．TABも1文字とする．この数値を超える文字は読み飛ばされる．

(4) LBLHSH

シンボルのハッシングブロックの数である．ハッシングについては「5章」で述べる．

(5) SRCSIZ

ソース読込バッファのバイト長であ

リスト 1-①

```
/****************************************/
/*  AS86のGLOBALな定数・変数・関数        */
/****************************************/

#define VERSION 1    /* 主バージョン番号 */
#define LEVEL   2    /* 副バージョン番号 */

#define IFMAX  16    /* IFの最大NESTING LEVEL */
#define SRCMAX 136       /* SOURCE PROGRAMの行BUFFER SIZE */
#define LBHSH  64    /* SYMBOLのHASH ENTRY数 */
#define LBLMAX 31    /* SYMBOLの最大文字数 */
#define SRCSIZ 0xFF00 /* SOURCE PROGRAMのBUFFERのSIZE */
#define OBJSIZ 1024      /* OBJECT PROGRAMのBUFFER SIZE */
#define LSTSIZ 1024      /* LISTINGのBUFFER SIZE */

#define ERROR -1         /* 一般的なERROR発生時の戻値 */
#define NORMAL 0         /* 正常終了時の戻値 */
#define TRUE  -1         /* 真理値TRUE */
#define FALSE  0         /* 真理値FALSE */

#define BYTMIN 0xFFFFFF80UL   /* 符号付きBYTE値の最小値 */
#define BYTMAX 0x0000007FUL   /* 符号付きBYTE値の最大値 */
#define WRDMIN 0xFFFF8000UL   /* 符号付きWORD値の最小値 */
#define WRDMAX 0x00007FFFUL   /* 符号付きWORD値の最大値 */
#define EXPTRU 0xFFFFFFFFUL   /* 式の真理値:TRUE */
#define EXPFLS 0x00000000UL   /* 式の真理値:FALSE */

#define DEF 0x8000       /* 定義済み */
#define REF 0x4000       /* 参照済み */
#define MLT 0x2000       /* 多重定義 */
#define SPC 0x0007       /* NUM～REGを分離するMASK */
#define NUM 0x0001       /* NUMBER */
#define LBL 0x0002       /* LABEL */
#define VAR 0x0003       /* VARIABLE */
#define REG 0x0004       /* REGISTER */

#define SIZNON 0x0000 /* NONE */
#define SIZBYT 0x0001 /* BYTE */
#define SIZWRD 0x0002 /* WORD */
#define SIZDWD 0x0004 /* DWORD */
#define SIZFWD 0x0006 /* FWORD */
#define SIZQWD 0x0008 /* QWORD */
#define SIZNER 0x0003 /* NEAR */
#define SIZFAR 0x0005 /* FAR */

#define BREG 0x0100      /* BYTE REGISTER 識別コード */
#define WREG 0x0200      /* WORD REGISTER 識別コード */
#define EREG 0x0300      /* 32BIT REGISTER 識別コード */
#define SREG 0x0400      /* SEGMENT REGISTER 識別コード */
#define CREG 0x0500      /* CONTROL REGISTER 識別コード */
#define DREG 0x0600      /* DEBUG REGISTER 識別コード */
#define TREG 0x0700      /* TEST REGISTER 識別コード */

#define TYPNUM 1         /* NUMBER */
#define TYPLBL 2         /* NEAR LABEL */
#define TYPVAR 3         /* VARIABLE */
#define TYPFAR 4         /* FAR LABEL */
#define TYPIND 5         /* REGISTER修飾(16 BIT) */
#define TYPINX 6         /* REGISTER修飾(32 BIT) */
#define TYPBRG 7         /* BYTE REGISTER */
#define TYPWRG 8         /* WORD REGISTER */
#define TYPSRG 9         /* CS・DS・ES・SS */
#define TYPSRX 10    /* FS・GS */
#define TYPERG 11    /* 32 BIT REGISTER */
#define TYPXRG 12    /* CRn・DRn・TRn */

struct lblt {        /* SYMBOL TABLEのDATA構造 */
    char             nam[LBLMAX+1];  /* SYMBOLの文字列 */
    unsigned int     att,            /* SYMBOLの属性 */
                     siz;            /* SYMBOLのSIZE */
    unsigned long p1val,                 /* PASS1の値 */
                  p2val;                 /* PASS2の値 */
};

struct opdv {        /* OPERANDの評価の結果(式の値)の形式 */
    unsigned int    att,             /* 式の属性 */
                    siz,             /* 式のSIZE */
                    bsr,             /* BASE REGISTER */
                    ixr,             /* INDEX REGISTER */
                    scl,             /* SCALING FACTOR */
                    ovr,             /* SEGMENT OVERRIDE */
                    aseg;        /* SEGMENTの絶対値 */
    unsigned long p1val,             /* PASS1の値 */
                  p2val;             /* PASS2の値 */
};

char
    lblbuf[LBLMAX+1], /* LABELの登録・参照用BUFFER */
    srcbuf[SRCMAX+1], /* SOURCE1行分のBUFFER */
    tknbuf[SRCMAX+1], /* TOKENのBUFFER */
    mchbuf[22],         /* 機械語表示用のBUFFER */
    locbuf[10],         /* ADDRESS表示用のBUFFER */
    stnbuf[6];          /* 行番号表示用のBUFFER */

int
    experr,             /* 式の評価中にERROR発生 */
    brackt,             /* []のNEST LEVEL */
    lstinh,             /* SOURCEのLISTINGへの一時出力禁止 */
    passno,             /* 処理中のPASS番号 */
    prctyp,             /* CPUの種別 */
    opgrp,              /* 命令の属するGROUP */
    opovr,              /* 命令に対するSEGMENT OVERRIDE */
    opasz,              /* 命令のADDRESS SIZE OVERRIDE */
    opcod,              /* 命令を識別するCODE */
    opmod,              /* mdregmem BYTE */
    opsib,              /* scixrbsr BYTE */
    prefix,             /* PREFIX命令のCODE */
    eopgm,              /* END命令を処理したときTRUE */
    ifcnt,              /* IFのNEST数 */
    ifflg[IFMAX];   /* 各LEVELのIFの真理値 */

unsigned long
    errcnt,             /* ERROR発生回数 */
    word,           /* SOURCE 1行で割り当てられたBYTE数 */
    lbls,           /* HASHING BLOCKのBYTE数 */
    lblb,           /* HASHING BLOCKのLABELの最大数 */
    opdisp,             /* OPERANDのDISPLACEMENT値 */
    lacmin,             /* 割り当てたMEMORY ADDRESSの最小値 */
    lacmax,             /* 割り当てたMEMORY ADDRESSの最大値+1 */
    lac;                /* LAC */

struct lblt
    _far * lbla[LBHSH];   /* SYMBOL HASHING BLOCKのENTRY ADDRESS */

struct opdv
    expval;             /* 式の値 */

/* FIO86.C */
int fioini(int argc, char *argv[]);
void srcopn(void);
void srccls(void);
int srcget(void);
void objopn(void);
void objput(int ch);
void objcls(void);
void objorg(unsigned long lc);
void mchopn(void);
void mchcls(void);
void mchbvst(int dt);
void mchwvst(int dt);
```


アセンブラ用語集
カウンタ　CXレジスタはおもに繰返し命令やシフト・ローテート命令の実行回数などのカウンタとして使われる．

る．ソース全体がこのバッファに入りきれば，各パスで再度ソースをディスクから読込むことはしない．

(6) OBJSIZ，LSTSIZ

オブジェクトプログラムとアセンブリリストをファイルに出力する際使用するファイル出力用バッファのバイト長である．

(7) ERROR，NORMAL

関数の戻値に使用し，その関数の処理状況を区別するのに使用する．ERRORは，処理に異常があったことを示し，NORMALは処理が無事終了したことを示す．

(8) TRUE，FALSE

関数の戻値や，フラッグに使用する変数の値として使用する．TRUEは，フラッグがONであることや検査結果が真であることを示す．FALSEは，フラッグがOFFであることや，検査結果が偽であることを示す．

(9) BYTMIN，BYTMAX

機械コードの1バイトで表せる符号付き数の最小値と最大値である．

(10)WRDMIN，WRDMAX

機械コードの1ワード（16ビット）で表せる符号付き数の最小値と最大値である．

(11)EXPTRU，EXPFLS

比較演算の結果を表す定数で，EXPTRUは比較の結果が真，EXPFLSは比較の結果が偽の場合の値である．

(12)DEF～REG

lblt構造体とopdv構造体のメンバattのビット位置を示す定数である．この中で，REFとMLTのビットはlblt構造体だけで，REGのビットはopdv構造体の中だけで使用される．

(13)SIZNON～SIZFAR

lblt構造体とopdv構造体のメンバsizに使用される簡易アセンブラの内部コードで，オペランドのサイズを表すのに使用される．

(14)BREG～TREG

簡易アセンブラがレジスタの種別を識別するために内部で使用するコードである．

(15)TYPNUM～TYPXRG

opdv構造体の内容から決まるオペランドのタイプを表す定数である．簡易アセンブラが機械コードを選択するときに使用する．

(16)lblt

シンボルテーブルのシンボル1つに対するデータの構造を定義している．メンバnamには，シンボルの綴がそのまま入る．このデータは¥0で終わる可変長の文字列データである．attは，シンボルの属性を表し，(12)で述べたDEF～VARのビットが使用される．p1valとp2valは，それぞれ簡易アセンブラがパス1とパス2で求めた値である．

(17)opdv

簡易アセンブラがオペランドである式を評価した結果を表すデータ構造である．attはp1valとp2valの属性であり，(12)のDEF～REGビットが使用される．ただし，REGビットが1のときには，値はメンバbsrにおかれる．メンバsizは，属性のLBLかVARが1のときのみ有効で，オペランドサイズを表す．メンバbsrとixrには，レジスタ修飾の式であるとき，式のベースレジスタとインデックスレジスタの内部コードが格納される．メンバsclは，式がスケールドインデックスのときにスケールコードが格納される．ovrは式の中にセグメントオーバライドが現れたとき，そのセグメントレジスタの内部コードが格納される．asegは，式の中にセグメントセレクタの絶対値が現れたとき，その値が格納される．p1valとp2valは，それぞれパス1とパス2のときのシンボルの値に基づく式の値である．

(18)lblbuf

シンボルの登録と参照を行うとき，この変数にシンボルの名前を設定する．

(19)srcbuf

ソース1行分のバッファで，srcget関数によりセットされる．

(20)tknbuf

ソースをtoken関数でトークンに分解した結果がこの変数にセットされる．

(21)mchbuf

アセンブリリストに機械コードを表示するためのバッファである．

(22)locbuf

アセンブリリストに機械コードのアドレスを表示するためのバッファである．

(23)stnbuf

アセンブリリストにソースの行番号を表示するためのバッファである．

アセンブラ用語集
間接アドレッシング　直接アドレッシングに対して，メモリアドレスの指定をレジスタを通じて行うものを間接アドレッシングという．

(24)experr

式の評価において，式が構文的におかしいときTRUEとなる．この変数がTRUEになると，式を評価する関数は直ちに処理を打ち切る．式を評価する関数は再帰的に呼び出されている可能性があるので，このような方法を使用して処理を打ち切っている．C処理系のライブラリにあるsetjmp/longjmpを使用すると効率的であろう．

(25)brackt

[]のネストを監視するための変数である.式の評価の途中[が現れると+1され,]が現れると-1される．式の中にレジスタが現れたとき，この変数が0でなければレジスタ修飾として，0であればレジスタ直接として処理される．なお，簡易アセンブラでは，[]の内部に[]を含むことはできない．

(26)lstinh

アセンブリリストを作成する際，機械コードの表示が2行以上にわたる場合，一時的に2行目以降にソースを表示しないために使用される．この変数の値がFALSEであればソースを表示し，TRUEであれば表示しない．

(27)passno

簡易アセンブラでは，すべてのパスでほとんどの関数を共用しているので，パスによる処理の差異がある場合，この変数を参照して実行すべき処理を決定している．この変数の値は，パス1のとき1，パス2のとき2，パス3のとき3にセットしなければいけない．

(28)prctyp

アセンブルするソースがどのCPUの種別に属するかを示す変数である．パスの初期化の際,0にセットされデフォルトの状態では8086である．この変数が変更されるのは,.186,.286,.386,.486および.8086のディレクティブが処理された場合で，それぞれ1，2，3，4および0にセットされる．

(29)opgrp

ソース行に含まれる命令に対応する命令テーブルに登録されているグループ番号がisopr関数によってセットされる．

(30)opovr

オペランドの内部にセグメントオーバーライドの記述があった場合，該当するセグメントオーバーライドの機械コードがmodgen関数によってセットされる．

(31)opasz

命令にアドレスサイズのプリフィックス命令が必要なとき，アドレスサイズプリフィックス命令の機械コードがmodgen関数によってセットされる．

(32)opcod

リスト1-2

```
void mchlvst(unsigned long dt);
void lstopn(void);
void lstputs(char *ptr, int typ);
void lstcls(void);
void lstout(void);
void lblout(void);
void asmerr(unsigned int code, char *admsg);
void errgen(unsigned int cod, char *admsg);
void ftlgen(unsigned int cod);

/* LBL86.C */
int lblini(void);
void lblreg(unsigned long vl, unsigned int at, unsigned int sz);
void lblev(char *pt);
struct lblt *lblref(char *lb);
char *token(char *pt);
int lblchk(char *lb);
void _far * alloc(unsigned long by);
void mfree(void _far * ba);

/* EXP86.C */
char *expr(char *pt);
int isreg(char *pt);

/* OP86.C */
int isopr(char *pt);
void srclin(void);
void ifop(char *pt);
char *fxop(char *pt);
int chktyp(struct opdv *xx);
void modgen(struct opdv *xx);
int segcnv(int sg);
int chkal(struct opdv *xx);
int chkcl(struct opdv *xx);
int chkax(struct opdv *xx);
int chkdx(struct opdv *xx);
int chkeax(struct opdv *xx);
int issht(char *pt);
int isend(char *pt);
char *nxtod(char *pt);
void m16(int mch, struct opdv *xx);
void n16(int mch, int ofs);
void n32(int mch, int ofs, int sg);
void mr16(int mch, int reg, struct opdv *xx);
void mr32(int mch, int reg, struct opdv *xx);
int typudl(struct opdv *xx);
int typudv(struct opdv *xx);
int typudf(struct opdv *xx);
int chkis8(struct opdv *xx);
int chkis16(struct opdv *xx);

/* PX86.C */
char *passx(char *pt, int cnt);
```


アセンブラ用語集
疑似命令　アセンブルされてマシンコードになるのではなく，セグメントの指定や生成するコードの種類などアセンブラに対しての指示を疑似命令という．

ソースに書かれている命令に対応する命令テーブルに登録されている命令識別コードがisopr関数によってセットされる.

⑶opmod

オペランドに対応する1-4-3で述べたmdregmemバイトがmodgen関数によってセットされる．機械コードの生成に使用する.

⑶opsib

オペランドに対応する1-4-3で述べたscixrbsrバイトがmodgen関数によってセットされる．機械コードの生成に使用する.

⑶prefix

プリフィックス命令を処理したとき，プリフィックス命令のコードが格納される．プリフィックス命令を前置された命令が正当性を調べるための変数だが，ディレクティブ以外ではチェックしていない.

⑶eopgm

ENDディレクティブを処理するとTRUEにセットされる変数である．各パスの処理を終了するために使用する.

⑶ifcnt

IF~ELSE~ENDIFブロックのネストレベルを示す変数である．IFディレクティブを処理すると+1され，ENDIFディレクティブを処理すると-1される．IF~ELSE~ENDIFブロックの外では0である.

⑶ifflg

IF~ELSE~ENDIFブロックのIF条件の真偽を保持する変数である．ifcntが0でないとき，現在のIF~ELSE~ENDIFブロックの真偽は，ifflg[ifcnt-1]に保持されている．この値は，IF条件が真のときTRUE，偽のときFALSEであり，真のときは通常のアセンブルが行われ，偽のときはELSE，ENDIF，ENDディレクティブ以外無視される.

⑶errcnt

エラーを検出した回数をカウントする．この変数はerrgen関数が呼び出される毎に+1される.

⑷word

各パスで，ソース1行が処理されると，その行で生成された機械コードのバイト長がセットされる．この変数は，mchbvst関数が呼び出されるたびに+1される．また，各行を処理する前にsrclin関数が0にリセットする.

⑷lbls

シンボルテーブルの各ハッシングブロックのバイト長である．lblini関数によってセットされアセンブル実行中は不変である.

⑷lblb

シンボルテーブルの1つのハッシングブロックに登録できるシンボル数である．lblini関数によってセットされアセンブル実行中は不変である.

⑷opdisp

「1-4-4」で説明した機械コードの変位の部分がmodgen関数によってセットされる．機械コードの生成に使用する.

⑷lacmin

機械コードを出力した最低のオフセットアドレスを記憶する．srclin関数によって設定され，objopn関数がオブジェクトファイル上に領域を確保するために使用する.

⑷lacmax

機械コードを出力した最高のオフセットアドレス+1を記憶する．srclin関数によって設定され，objopn関数がオブジェクトファイル上に領域を確保するために使用する.

⑷lac

ソース各行で生成される機械コードの先頭アドレスを保持する変数である．この変数は，ORGディレクティブの処理中に直接セットされる．これ以外の場合，各行の処理中は変化しない．各行の処理が終了後，srclin関数が元の値にword変数の内容を加えることにより更新する.

⑷lbla

シンボルテーブルの各ハッシングブロックの先頭位置を保持するポインタの配列である．lblini関数によりセットされ，簡易アセンブラが動作中は変化しない.

⑷expval

expr関数が1つのオペランドを評価した結果をセットするopdv型構造体である.

⑷fioini

main関数の引数をそのまま渡して，コマンドラインの処理を行う．その結果を元に，ソースファイル名，オブジェクトファイル名，アセンブリリストのファイル名と各種スイッチのセットを行う．ソースファイルの入力バッファの確保もこの関数で行う.

(50) srcopn, srccls, srcget

これらの関数は，ソースファイルを操作する関数である．srcopn関数は，ソースファイルをオープンしてソースファイルをバッファに読込む．最初の呼び出しでソースファイルがバッファに入りきれば，2度目の呼び出しからはファイルのオープンはせず，バッファからの読出の準備をするだけである．srccls関数は，ソースファイルをクローズする．ソースファイルがバッファに入りきれば，2度目の呼び出しからは実際のファイルクローズは行わない．srcget関数は，ソース1行分をsrcbufに読込む関数である．ソース1行分が正常に読込めればNORMALを返し，EOFになったならERRORを返す．ファイル操作にエラーが発生した場合は，asmerr関数を経由してアボートする．

(51) objopn, objput, objcls, objorg

これらの関数は，オブジェクトファイルを操作する関数である．objopn関数は，オブジェクトファイルをオープンする．そして，lacmax-lacminバイトの0のデータを書込んでオブジェクトの出力の準備を行う．オブジェクトファイルの先頭の機械コードは，lacmin番地の機械コードであって，ソースファイルの先頭の機械コードとは限らない．objputは，オブジェクトファイルに1バイト書込む関数であり，objclsは，オブジェクトファイルをクローズする関数である．objorg関数は，ORGディレクティブから呼び出され，オペランドで指定された位置に次の機械コードが出力されるようファイルポインタを移動させる．これらの関数は，passno =3でない限りは，無操作正常終了する．また，ファイル操作で異常が生じた場合は，asmerr関数を経由してアボートする．

(52) mchopn, mchcls, mchbvst, mchwvst, mchlvst

これらの関数は，機械コードをオブジェクトに出力すると同時にアセンブリリストにオブジェクトを表示するために使用される．mchopnは，1行分の表示に先立ち，表示の初期化のために呼ばれる．mchclsは，1行分の表示が終了した時点で呼び出され，表示の後始末を行う．mchbvstは1バイト，mchwvstは1ワード，mchlvstは2ワードの機械コードをそれぞれ表示し，同時にobjput関数を呼び出してオブジェクトファイルに機械コードを出力する．passno =3でない限り，mchbvst，mchwvstおよびmchlvstの各関数はwordをカウントアップするだけであり，mchopnとmchcls関数は正常終了する．

(53) lstopn, lstputs, lstcls, lstout, lblout

これらの関数は，アセンブリリストを出力するための関数である．lstopnとlstclsはアセンブリリスト用のファイルをそれぞれオープン・クローズする．lstputsは，1行分の文字列データをアセンブリリスト用のファイルに出力する．このとき文字列データの最後に¥rと¥nは不要である．行番号と1行中の文字数の管理を行っていて，必要であれば行替えやページ替えを行う．lstoutは，機械コードのアドレスと機械コードを含めたソース1行分のリストを出力する．lbloutは，シンボルテーブルを名前の昇順にソートし，シンボルの値と属性と共にシンボルの一覧表をアセンブリリスト用のファイルに出力する．これらの関数は，passno =3でない限り無操作正常終了となる．アセンブリリスト用のファイルを操作中にエラーが発生したときは，asmerr関数を経由してアボートする．

(54) asmerr

エラーメッセージを標準出力装置に表示して，簡易アセンブラをアボートする．

(55) errgen

ソースプログラムに問題があるとき，エラーメッセージを標準出力装置とアセンブリリスト用のファイルに出力する．メッセージはMASMと同じ形式で，多くのエディタでタグジャンプに使用できる．

(56) fltgen

ソースプログラムの中に致命的なエラーを発見したとき，エラーメッセージを表示して簡易アセンブラをアボートする．致命的なエラーは，IFとORGディレクティブでオペランドの中に前方参照を含んだり，オペランドのタイプが間違っていた場合に発生し，パス1の処理中にだけ検出される．

(57) lblini

シンボルテーブルの領域確保と初期化を行う関数である．正常に領域確保と初期化が行われたときはNORMALを返し，領域確保ができなかったときはERRORを返す．

(58) lblreg

lblbufに格納されているシンボルの登録を行う．パス3のときはフェーズエラーを検出することだけを行い，登録は行わない．

(59) lblev

lblbufに格納されているシンボル

アセンブラ用語集
クオッドワード　64ビット（8バイト）で表せる値をクオッドワードといい，倍精度浮動小数点などを表すのに使われる．

の値を求めて，expvalにセットする．この関数はexpr関数から間接的に呼び出されて，式の評価に使用される．

(60)lblref

lblbufに格納されているシンボルの登録位置を求めるための関数である．この位置は，未登録のシンボルのときは登録すべき位置であり，登録済みのシンボルのときは登録されている位置である．テーブルが一杯のときはヌルポインタを返す．

(61)token

アセンブラのソースをトークンに分解する．トークンとは，処理の単位となる文字列または記号で，簡易アセンブラのトークンには，シンボル，命令，文字列，演算子，数値，区切り記号がある．

(62)lblchk

シンボルとして有効な文字列かどうかを調べる．有効な文字列であれば，NORMALが無効な文字列であればERRORが返される．無効と判定されるのは，文字列が数字で始まっている場合と，予約語と同じスペルである場合である．簡易アセンブラの予約語は，プロセッサ命令，ディレクティブおよびプログラムLBL86.Cにあるrsvwrdと大文字小文字の区別なく一致するものである．

(63)alloc

インラインアセンブラで記述したDOSから直接メモリを獲得する関数である．ライブラリ関数のhalloc関数でも，メモリの割当は可能だが，DOSから残りメモリの量を調べる関数rembytを使用してメモリの残量を確認した上で，メモリを割り当てる方式を採用しているので，一貫性を持たせるためにこの関数を使用している．

(64)mfree

これもインラインアセンブラで記述した関数で，DOSに直接メモリを返却する．alloc関数で割り当てたメモリは，この関数で返却しなければならない．

(65)expr

オペランドすなわち式の記述を解析して式の値を求める（評価する）関数である．値は，expvalにセットされる．詳細は「7章」で述べる．

(66)isreg

引数のポインタが示す文字列がレジスタのときTRUEを返し，同時にその値をexpvalにセットする関数である．レジスタでないときはFALSEを返す．

(67)isopr

引数のポインタが示す文字列がプロセッサ命令かディレクティブのときTRUEを返し，opcodとopgrpにプログラムOP86.Cにある命令テーブルoplstに登録されている値をセットする．そうでないときはFALSEを返す．

(68)srclin

srcbufに格納されている1行分のソースのパス処理を行う関数である．ソースの構文解析を行ってシンボル，命令およびオペランドの処理を行う関数を呼び出す．

(69)ifop

srclinから呼び出される関数で，IF～ELSE～ENDブロックの真偽により命令を処理するかしないかを決定する関数である．

(70)fxop

プリフィックス命令を処理する関数から呼び出されて，命令コードの処理を継続させるための関数である．

(71)chktyp

オペランドの値の組み合わせにより，オペランドのタイプを判定してその結果をTYPNUM～TYPXRGのコードにより返す関数である．

(72)modgen

オペランドの値の組み合わせから，mdregmemバイト，scixrbsrバイト，変位，アドレスサイズプリフィックス命令コードおよびセグメントオーバーライド命令コードを作成し，それぞれopmod，opsib，opdisp，opaszおよびopovrに格納する．

(73)segcnv

セグメントレジスタの内部コードをセグメントオーバーライド命令コードに変換して返す．

(74)chkal，chkcl，chkax，chkdx，chkeax

これらの関数は，オペランドの値がそれぞれAL，CL，AX，DXまたはEAXレジスタであるときTRUEを返し，そうでないときFALSEを返す．

(75)issht

ポインタ引数で示される場所にSHORT演算子があるときTRUEを返し，そうでないときFALSEを返す関数である．

(76)isend

アセンブラ用語集
構造体　必ずしも同一の型ではない複数のメンバが集合したデータ構造を指す．

ポインタ引数で示される場所がオペランドの終わりの位置であればTRUEを返し，そうでなければFALSEを返す関数である．

(77)nxtod

ポインタ引数で示される位置にコンマがあることを確認して，次のオペランドの頭出しをする関数である．2オペランド以上のオペランドのある命令の処理関数から呼び出される．

(78)m16

MOV AX, I16命令などの形式の機械コードをオブジェクトファイルへ出力する関数である．

(79)n16

CALL d16/JMP d16命令の形式の機械コードをオブジェクトファイルへ出力する関数である．

(80)n32

CALL I32/JMP I32命令の形式の機械コードをオブジェクトファイルへ出力する関数である．

(81)mr16, mr32

mdregmemバイトの必要な命令の機械コードをmodgenの作成したコードを利用してオブジェクトファイルへ出力する関数である．mr16は286以下のCPU用，mr32は386以上のCPU用である．

(82)typudl, typudv, typudf

これらの関数は，通常はオペランドのタイプを返すが，パス1で前方参照のシンボルを含む式に出会ったときにはデフォルトの属性を返す関数である．デフォルトの属性は，機械コードが最も長くなるようなものが選ばれる．

(83)chkis8, chkis16

これらの関数は，オペランドの変位の部分がchkis8では符号付き8ビット，chkis16では符号付き16ビットの範囲に入っていればTRUEを返し，そうでなければFALSEを返す関数である．chkis16は，将来の拡張用の関数であり未使用である．

(84)passx

opgrpの値によりop01～op63の関数に処理を切りわけて機械コードの生成処理を行う関数である．

プログラム2 (AS86.C)

プログラム2 (AS86.C)には，アセンブラの入り口であるmain関数がある．main関数は，コマンドラインの引数を受け取るが，環境変数は使用しな

リスト 2-1

```
/****************************************/
/*  アセンブラ全体の制御部               */
/****************************************/

#include "AS86.H"
#include <STDLIB.H>
#include <STDIO.H>

static void pass1(void);
static void pass2(void);
static int pass3(void);
static void pasini(void);

/* MAIN関数 */
int main(int argc, char *argv[]) {

    if(fioini(argc, argv)!=NORMAL) exit(1);
    if(lblini()!=NORMAL) exit(1);
    pass1();
    pass2();
    return pass3();
}

/* PASS 1の実行 */
static void pass1(void) {

    printf("80x86 Assembler (V%02d.%02d)", VERSION, LEVEL);
    printf(", Programmer's Page, May, 1992.¥n");
    passno=1;
    pasini();
    srcopn();
    while(1) {
        if(srcget()!=NORMAL) break;
        srclin();
        if(eopgm) break;
    }
    srccls();
    printf("Pass 1 End.¥n");
}

/* PASS 2の実行 */
static void pass2(void) {

    passno=2;
    pasini();
    srcopn();
    while(1) {
        if(srcget()!=NORMAL) break;
        srclin();
        if(eopgm) break;
    }
    srccls();
    printf("Pass 2 End.¥n");
}

/* PASS 3の実行 */
static int pass3(void) {

    char    buf[80];
    int     ret;

    passno=3;
    objopn();
    pasini();
    srcopn();
    lstopn();
    while(1) {
        if(srcget()!=NORMAL) break;
        srclin();
        if(eopgm) break;
```


アセンブラ用語集
コードセグメント　プログラムの命令コードが格納されるアドレスのセグメント．

リスト 2-2

```
	}
	srccls();
	objcls();
	printf("Pass 3 End.¥n");
	lblout();
	buf[0]=(char)'¥0';
	lstputs(buf, 1);
	if(errcnt) {
		sprintf(buf, "%ld Error(s). Assembly End.", errcnt);
		ret=1;
	}
	else {
		sprintf(buf, "No Error. Assembly End.");
		ret=0;
	}
	lstputs(buf, 1);
	printf("%s", buf);
	lstcls();
	return ret;
}

/* 各PASSの初期設定 */
static void pasini(void) {

	eopgm=FALSE;
	lac=0L;
	errcnt=0L;
	prctyp=0;
	ifcnt=0;
	if(passno!=3) {
		lacmax=0UL;
		lacmin=0xFFFFFFFFUL;
	}
}

```

いので無視する．main関数では，最初にfioini関数を呼び出してコマンドラインの解析，ソースファイル，オブジェクトファイル，アセンブリリスト用ファイルの入出力の初期化，入出力バッファの確保を行う．次にlblini関数を呼び出し，シンボルテーブル領域の確保，シンボルテーブルのエントリアドレスの設定，シンボルテーブルのクリアの処理を行う．

これら２つの関数は，問題が発生すると定数NORMAL以外の値を返す．もし，NORMAL以外のが返ってきたら，アセンブル処理はできないので終了コード１で処理を異常終了する．次にpass1, pass2およびpass3の関数を順次呼び出す．これらの関数は，コマンドラインで指定されたソースプログラムに対してパス１，パス２およびパス３を実行する関数である．pass3には戻値がある．パス３の戻値は，ソースプログラムをアセンブル中にエラーを検出したとき１，エラーがなかった場合0である．パス３が終了するとpass3関数の戻値を終了コードとしてアセンブラのすべての処理を終了する．

簡易アセンブラでは，各パスの処理はすべて同じ関数を使用して行っている．その基本的な手順をＣ言語風の疑似プログラムで示すと，図４のようになる．

パスの処理は，ソースの最初の行からEND命令が処理されるか，ソースファイルがEOFになるまで続けられる．パス３ではとくにオブジェクトプログラムとリストを出力するので，これらのファイルをオープン，クローズしなければならない．またシンボルリストの出力も必要である．

各パスの最初に行う変数の初期化は，passno，eopgm，lac，prctyp，ifcnt，lacmaxおよびlacminであり，passnoを除いてpasini関数で行われる．この初期値がアセンブルするプログラムの先頭でのデフォルトの値となる．すなわち，ORGディレクティブがなければ0000H番地から機械コードを出力し，アセンブル対象のCPUは8086であり，IF~ELSE~ENDIFのブロックの範囲外であるということである．lacminおよびlacmaxは，objopn関数で使用されるので，パス３のときは初期化は不要である．

図４

```
各パスの処理(void) {
	パスの番号をpassnoにセットする;
	共通変数の初期化を行う;
	ソースファイルをオープンする;
	while(1) {
		ソースを1行読込む;
		if(ソースがEOF) break;
		ソースを1行処理する;
		if(ソースの命令がEND命令) break;
	}
	ソースファイルをクローズする;
```

アセンブラ用語集
コプロセッサ　特定の命令を主プロセッサに代わって高速に実行するプロセッサ．数値演算コプロセッサやグラフィックプロセッサなどがある．

4 入出力

プログラム3（FIO86.C）には，簡易アセンブラの入出力に関係する関数が集中している．図3で見たとおり，アセンブラが入力するのはソースプログラムファイルであり，出力するのはオブジェクトプログラムファイルとアセンブリリストファイルである．アセンブリリストファイルは，さらにソースリストとシンボルリストに分かれている．プログラム3の先頭には，コマンドラインの処理が含まれているので，まず簡易アセンブラのコマンドラインから説明しよう．

リスト3　FIO86.C

```
/****************************************/
/*  入出力部(AS86)                       */
/****************************************/

#include <STDLIB.H>
#include <STDIO.H>
#include <STRING.H>
#include <FCNTL.H>
#include <SYS¥TYPES.H>
#include <SYS¥STAT.H>
#include <IO.H>
#include "AS86.H"

static char
    delim1[4] = " ¥t,",         /* COMMANDの読み飛ばし文字 */
    delim2[6] = " ¥t,/",        /* COMMANDの区切り文字 */
    linbuf[SRCMAX+40],          /* LIST 1行のBUFFER */
    objbuf[OBJSIZ],             /* OBJECTのFILE BUFFER */
    lstbuf[LSTSIZ],             /* LISTING FILEのBUFFER */
    srcfn[66],                  /* SOURCE PROGRAMのFULL PATH名 */
    objfn[66],                  /* OBJECT PROGRAMのFULL PATH名 */
    lstfn[66],                  /* LISTING FILEのFULL PATH名 */
    _far * srctop,              /* SOURCE PROGRAMのMEMORY BUFFER */
    _far * srcpnt,              /* SOURCE PROGRAMのBUFFER読出POINTER */
    *objpnt,                    /* OBJECT PROGRAMのBUFFER書込POINTER */
    *lstpnt;                    /* LISTINGのBUFFER書込POINTER */

static int
    lstsw,      /* LISTING FILE作成SWITCH */
    lblsw,      /* LABEL LIST作成SWITCH */
    objsw,      /* OBJECT PROGRAM作成SWITCH */
    pagsw,      /* 1page当たりの行数 */
    widsw,      /* 1行当たりの文字数 */
    srchnd,     /* SOURCE FILEのHANDLE */
    objhnd,     /* OBJECT FILEのHANDLE */
    lsthnd,     /* LISTING FILEのHANDLE */
    objcnt,     /* OBJECT PROGRAMのBUFFER中のBYTE COUNT */
    lstcnt,     /* LISTINGのBUFFER中のBYTE COUNT */
    srceof,     /* SOURCE PROGRAMのEOF MARK */
    srcbig,     /* SOURCE PROGRAMがBUFFERに入切らない */
    linno,      /* 1 PAGE中の行番号 */
    pagno,      /* LISTINGのPAGE番号 */
    mchlcn,     /* MCHBUFに表示したBYTE数 */
    skreq;      /* OBJECTの次のDATAの出力でSEEK必要 */

static unsigned int
    srcnum,     /* SOURCE PROGRAMのBUFFER中のBYTE COUNT */
    srccnt;     /* SOURCE PROGRAMのBUFFER中の残りBYTE COUNT */

static unsigned long
    stncnt,     /* 行番号COUNTER */
    laclst;     /* LISTING用LAC */

static long
    skpos;      /* OBJECT FILEのSEEK位置 */

static int _near cmdanl(char *src);         /* COMMAND LINEの解析 */
static void _near fioset(void);             /* DEFAULT FILE名 */
static int _near srcset(void);              /* SOURCE 1行の読込 */
static void mchlst(int dt);                 /* 機械語の表示 */
static void _near lstput(int ch);   /* LIST 1文字の出力 */
static void _near lstttl(int typ);          /* PAGE見出しの出力 */
static void _near lblsrt(void);             /* LABELのSORT */
static char * _near fnset(char *buf,char *nam);   /* FILE名の解析 */
static void _near strrpt(char *ptr, int chr, int num);
                /* 文字を繰り返して文字列を作成 */

/* 入出力の初期化入り口 */
int fioini(int argc, char *argv[]) {

    int     ii,
            flg;

    srcbuf[0]=(char)'¥0';
    ii=1;
    while(1) {              /* argvを1つの文字列へ */
        if(ii==argc) break;
        strcat(srcbuf, argv[ii]);
        ii++;
        strcat(srcbuf, " ");
    }
    srcfn[0]=(char)'¥0';            /* FILE名とSWITCHの初期値 */
    objfn[0]=(char)'¥0';
    lstfn[0]=(char)'¥0';
    lstsw=FALSE;
    objsw=FALSE;
    lblsw=TRUE;
    pagsw=66;
    widsw=132;
    flg=cmdanl(srcbuf);
    if(flg!=NORMAL) return flg;
    fioset();
```

アセンブラ用語集
サインフラグ　演算の結果で，最上位ビットが1の時セットされ，0の時リセットされる．正負の判断に使われるフラグ．

```
    srcbig=TRUE;
    return NORMAL;
}

/* COMMAND LINEの解析 */
static int _near cmdanl(char *src) {

    char    *dst,
            fx[66],
            drv[4],
            dir[66],
            fn[10],
            ft[6];
    int     chr,
            fcnt;
    long    num;

    fcnt=0;
    src=srcbuf;
    while(1) {
        src=src+strspn(src, delim1);
        chr=(int)*src++;
        if(chr==0) break;
        if(chr=='/') {      /* SWITCH指定の解析 */
            chr=(int)*src++;
            if(chr=='L' || chr=='l') {
                chr=(int)*src;
                if(strchr(delim2, chr)!=NULL)
                    lstsw=TRUE;
                else goto L1;
            }
            else if(chr=='N' || chr=='n') {
                chr=(int)*src;
                if(strchr(delim2, chr)!=NULL)
                    lblsw=FALSE;
                else goto L1;
            }
            else if(chr=='O' || chr=='o') {
                chr=(int)*src;
                if(strchr(delim2, chr)!=NULL)
                    objsw=TRUE;
                else goto L1;
            }
            else if(chr=='P' || chr=='p') {
                num=strtol(src, &src, 10);
                chr=(int)*src;
                if(strchr(delim2, chr)==NULL) goto L1;
                if(num>=30 && num<=256) pagsw=(int)num;
            }
            else if(chr=='W' || chr=='w') {
                num=strtol(src, &src, 10);
                chr=(int)*src;
                if(strchr(delim2, chr)==NULL) goto L1;
                if(num>=80 && num<=256) widsw=(int)num;
            }
            else goto L1;
        }
        else {          /* ファイル指定の解析 */
            src--;
            if(fcnt>2) {
                asmerr(2, src);
                return ERROR; /* ファイル数過多 */
            }
            src=fnset(src, fx);
            chr=(int)*src;
            if(strchr(delim2, chr)==NULL) {
                asmerr(3, src);
                return ERROR; /* ファイル指定不良 */
            }
            _splitpath(fx, drv, dir, fn, ft);
            if((int)ft[0]=='¥0') {
                if(fcnt==0) strcpy(ft, ".ASM");
                else if(fcnt==1) strcpy(ft, ".COM");
                else strcpy(ft, ".LST");
            }
            _makepath(fx, drv, dir, fn, ft);
            if(fcnt==0) dst=srcfn;
            else if(fcnt==1) dst=objfn;
            else dst=lstfn;
            _fullpath(dst, fx, 65);
            fcnt++;
        }
    }
    return NORMAL;
L1:
    asmerr(1, src);
    return ERROR;           /* SWITCH指定不良 */
}

/* FILE指定のCHECKとBUFFER確保 */
static void _near fioset(void) {

    char    drv[4],
            dir[66],
            fn[10],
            ft[6];

    if((int)srcfn[0]=='¥0') asmerr(4, srcbuf); /* SOURCE指定なし */
    _splitpath(srcfn, drv, dir, fn, ft);
    if(objsw!=0 && (int)objfn[0]=='¥0') {
        strcpy(ft, ".COM");
        _makepath(objfn, drv, dir, fn, ft);
    }
    if(lstsw!=0 && (int)lstfn[0]=='¥0') {
        strcpy(ft, ".LST");
        _makepath(lstfn, drv, dir, fn, ft);
    }
    srctop=(char *)alloc((unsigned long)SRCSIZ);
    if(srctop==NULL) asmerr(0, "ソースバッファ"); /* メモリ不足 */
}

/* SOURCE FILEのOPEN */
void srcopn(void) {

    if(srcbig) {    /* SOURCEがバッファより大きい */
        srchnd=open(srcfn, O_BINARY|O_RDONLY);
        if(srchnd==-1)    asmerr(5, srcfn);
        srcnum=read(srchnd, srctop, SRCSIZ);
        if(srcnum==-1) asmerr(6, srcfn);
        if(srcnum==SRCSIZ) {
            srcbig=TRUE;
            srceof=FALSE;
        }
        else {
            srcbig=FALSE;
            srceof=TRUE;
            close(srchnd);
        }
    }
    srcpnt=srctop;
    srccnt=srcnum;
    stncnt=0;
}

/* SOURCE FILEのCLOSE */
void srccls(void) {

    if(srcbig!=FALSE) close(srchnd);
}

/* SOURCE PROGRAMをSRCBUFへ取得する入り口 */
int srcget(void) {

    int     flg;

    flg=srcset();
    stncnt++;
    sprintf(stnbuf, "%5ld", stncnt);
    return flg;
}

/* 1行分のSOURCE PROGRAMをSRCBUFへ転送 */
static int _near srcset(void) {

    char    *dst;
    int     chr,
            cnt;

    dst=srcbuf;
    cnt=0;
    while(1) {
        if(srccnt==0) {
            if(srcbig==FALSE || srceof!=FALSE) return ERROR;
            srccnt=read(srchnd, srctop, SRCSIZ);
            if(srccnt==ERROR) asmerr(6, srcfn);
            if(srccnt!=SRCSIZ) srceof=TRUE;
            srcpnt=srctop;
            continue;   /* SOURCEをさらに読込んだ */
        }
        chr=(int)*srcpnt++;
        srccnt--;
        if(chr=='¥x1A') {
            srceof=TRUE;
            *dst=(char)'¥0';
            return ERROR; /* SOURCEがEOFになった */
        }
        if(chr=='¥n') {
            *dst++=(char)'¥0';
            break;          /* 1行srcbufへ転送完了 */
        }
        if(cnt<SRCMAX && (chr>=0x20 || chr=='¥t')) {
            *dst++=(char)chr;
            cnt++;
        }
    }
    return NORMAL;
}

/* OBJECT FILEのOPEN */
void objopn(void) {

    int             flg,
                    qlen;
    unsigned long cnt;

    if((int)objfn[0]) {
        objhnd=open(objfn, O_BINARY|O_CREAT|O_TRUNC|O_WRONLY,
            S_IWRITE|S_IREAD);
        if(objhnd==ERROR) asmerr(7, objfn);
        memset(objbuf, 0, OBJSIZ);
        if(lacmax>lacmin) cnt=lacmax-lacmin;
        else cnt=0L;
        while(cnt) {
            if(cnt>(unsigned long)OBJSIZ) qlen=OBJSIZ;
            else qlen=(int)cnt;
            flg=write(objhnd, objbuf, qlen);
            if(flg==ERROR)    asmerr(8, objfn);
            if(flg!=qlen) asmerr(9, objfn);
            cnt-=(unsigned long)qlen;
        }
        lseek(objhnd, 0L, SEEK_SET);
    }
    objpnt=objbuf;
    objcnt=0;
    skreq=0;
```

```
}

/* OBJECT FILEへ1BYTE出力 */
void objput(int ch) {

    int     flg;
    long    chk;

    if((int)objfn[0]=='¥0') return;
    if(passno!=3) return;
    if(skreq) {
        if(objcnt) {
            flg=write(objhnd, objbuf, objcnt);
            if(flg==ERROR)    asmerr(8, objfn);
            if(flg!=objcnt) asmerr(9, objfn);
            objpnt=objbuf;
            objcnt=0;
        }
        chk=lseek(objhnd, skpos, SEEK_SET);
        if(chk==-1L) asmerr(11, objfn);
        skreq=FALSE;
    }
    *objpnt++ =(char)ch;
    objcnt++;
    if(objcnt==OBJSIZ) {
        flg=write(objhnd, objbuf, OBJSIZ);
        if(flg==ERROR)    asmerr(8, objfn);
        if(flg!=OBJSIZ) asmerr(9, objfn);
        objpnt=objbuf;
        objcnt=0;
    }
}

/* OBJECT FILEのCLOSE */
void objcls(void) {

    int     flg;

    if((int)objfn[0]=='¥0') return;
    if(passno!=3) return;
    if(objcnt) {
        flg=write(objhnd, objbuf, objcnt);
        if(flg==ERROR) asmerr(8, objfn);
        if(flg!=objcnt) asmerr(9, objfn);
    }
    flg=close(objhnd);
    if(flg!=NORMAL) asmerr(8, objfn);
}

/* OBJECT FILEの出力位置変更(ORG命令) */
void objorg(unsigned long lc) {

    if((int)objfn[0]=='¥0') return;
    if(passno!=3) return;
    skpos=(long)lc-lacmin;
    skreq=TRUE;
}

/* 1行分のOBJECT作成開始処理 */
void mchopn(void) {

    laclst=lac;
    sprintf(locbuf, "%04X    ", laclst);
    mchlcn=0;
    lstinh=FALSE;
    mchbuf[0]=(char)'¥0';
}

/* OBJECTのLISTING用データの作成 */
static void mchlst(int dt) {

    char    buf[6];

    dt&=0xFF;
    sprintf(buf, "%02X", dt);
    strcat(mchbuf, buf);
    mchlcn++;
    if(mchlcn==10) {
        lstout();
        laclst+=10L;
        mchlcn=0;
        lstinh=TRUE;
        mchbuf[0]=(char)'¥0';
        sprintf(locbuf, "%04X    ", laclst);
    }
}

/* 1行分のOBJECT出力終了処理 */
void mchcls(void) {

    if(mchlcn) lstout();
    lstinh=FALSE;
    if(word==0L) lstout();
}

/* BYTE DATAをOBJECTとして出力 */
void mchbvst(int dt) {

    dt&=0x00FF;
    mchlst(dt);
    objput(dt);
    word++;
}

/* WORD DATAをOBJECTとして出力 */
void mchwvst(int dt) {

    mchbvst(dt);
    mchbvst(dt>>8);
}

/* LONG DATAをOBJECTとして出力 */
void mchlvst(unsigned long dt) {

    int     di;

    di=(int)(dt);
    mchbvst(di);
    dt>>=8;
    di=(int)(dt);
    mchbvst(di);
    dt>>=8;
    di=(int)(dt);
    mchbvst(di);
    dt>>=8;
    di=(int)dt;
    mchbvst(di);
}

/* LISTING FILEのOPEN */
void lstopn(void) {

    if((int)lstfn[0]) {
        lsthnd=open(lstfn, O_BINARY|O_CREAT|O_TRUNC|O_WRONLY,
            S_IWRITE|S_IREAD);
        if(lsthnd==ERROR) asmerr(7, lstfn);
    }
    lstpnt=lstbuf;
    lstcnt=0;
    pagno=0;
}

/* LISTING FILEへ1文字出力 */
static void _near lstput(int ch) {

    int     flg;

    if((int)lstfn[0]=='¥0') return;
    *lstpnt++ =(char)ch;
    lstcnt++;
    if(lstcnt==LSTSIZ) {
        flg=write(lsthnd, lstbuf, LSTSIZ);
        if(flg==ERROR) asmerr(8, lstfn);
        if(flg!=OBJSIZ) asmerr(9, lstfn);
        lstpnt=lstbuf;
        lstcnt=0;
    }
}

/* LISTING FILEへ文字列を出力 */
void lstputs(char *ptr, int typ) {

    int     chr,
            col,
            fld;

    if(pagno==0) lstttl(typ);
    else if(linno==0) lstttl(typ);
    col=0;
    while(1) {
        chr=(int)*ptr++;
        if(chr=='¥0') break;
        if(chr=='¥t') col=(col&0xFFF8)+8;
        if(col>=widsw) {
            lstput('¥r');
            lstput('¥n');
            linno--;
            if(linno==0) lstttl(typ);
            col=0;
            fld=40;
            if(typ) fld=50;
            while(fld>0) {
                lstput(' ');
                col++;
                fld--;
            }
            if(chr=='¥t') col=(col&0xFFF8)+8;
        }
        lstput(chr);
        col++;
    }
    lstput('¥r');
    lstput('¥n');
    linno--;
}

/* LISTING FILEのCLOSE */
void lstcls(void) {

    int     flg;

    if((int)lstfn[0]=='¥0') return;
    if(lstcnt) {
        flg=write(lsthnd, lstbuf, lstcnt);
        if(flg==ERROR) asmerr(8, lstfn);
        if(flg!=lstcnt) asmerr(9, lstfn);
    }
    flg=close(lsthnd);
    if(flg!=NORMAL) asmerr(8, lstfn);
}

/* LISTの改ページとTITLEの作成 */
static void _near lstttl(int typ) {

    char    *ptr,
            buf[258];
    int     wx;
```


アセンブラ用語集
システムコール　ソフトウェア割り込みのこと．CPUはこの命令を受け付けると，割り込みポインタアドレスを参照して，そのアドレスに制御を移す．

```
    buf[0]=(char)'¥0';
    if(pagno) lstput('¥f');
    pagno++;
    linno=pagsw;
    sprintf(buf, "AS86 ASSEMBLER  (V%02d.%02d)  ", VERSION, LEVEL);
    wx=41+strlen(srcfn);
    if(widsw>=wx) {
        strcat(buf, srcfn);
        wx=strlen(buf);
        ptr=buf+wx;
        wx=widsw-wx-12;
        strrpt(ptr, ' ', wx);
        ptr=buf+strlen(buf);
        sprintf(ptr, "  PAGE %5d", pagno);
        lstputs(buf, typ);
        buf[0]=(char)'¥0';
        lstputs(buf, typ);
    }
    else {
        wx=widsw-41;
        ptr=buf+29;
        strrpt(ptr, ' ', wx);
        ptr=buf+strlen(buf);
        sprintf(ptr, "  PAGE %5d", pagno);
        lstputs(buf, typ);
        wx=(widsw-strlen(srcfn))/2;
        strrpt(buf, ' ', wx);
        strcat(buf, srcfn);
        lstputs(buf, typ);
        buf[0]=(char)'¥0';
    }
    lstputs(buf, typ);
    if(typ==0) {
        strcpy(buf, " Location Object                    Stno.");
        strcat(buf, "  Source Statement");
    }
    else strcpy(buf, " Label
        "Value   Attrib.");
    lstputs(buf, typ);
    buf[0]=(char)'¥0';
    lstputs(buf, typ);
}
/* SOURCE PROGRAMのLIST作成入り口 */
void lstout(void) {
    char    *ptr;
    if(passno!=3) return;
    sprintf(linbuf, " %-9s%-21s", locbuf, mchbuf);
    if(lstinh==FALSE) {
        ptr=linbuf+strlen(linbuf);
        sprintf(ptr, "%-8s", stnbuf);
        strcat(linbuf, srcbuf);
    }
    lstputs(linbuf, 0);
}
/* SYMBOL TABLEのLIST作成入り口 */
void lblout(void) {
    char        buf[40],
                *atr;
    int         cnt,
                blk,
                wrk;
    struct lblt   *pos;
    if((int)lstfn[0]=='¥0' || lblsw==FALSE) return;
    lblsrt();
    cnt=0;
    blk=0;
    pos=lbla[0];
    if((int)pos->nam[0]!='¥0') {
        lsttl(1);
        while(1) {
            if(cnt>=(int)lblb) {
                blk++;
                if(blk>=LBHSH) break;
                pos=lbla[blk];
                cnt=0;
            }
            if((int)pos->nam[0]=='¥0') break;
            if(linno==0) lsttl(1);
            sprintf(linbuf, "  %-*s ", LBLMAX+1, pos->nam);
            wrk=(int)pos->att;
            sprintf(buf, "%081X ", pos->p2val);
            strcat(linbuf, buf);
            if((wrk&DEF)==0) atr="UDF";
            else if((wrk&MLT)!=0) atr="MLT";
            else if((wrk&SPC)==NUM) atr="NUM";
            else if((wrk&SPC)==LBL) atr="LBL";
            else if((wrk&SPC)==VAR) atr="VAR";
            else atr="   ";
            strcat(linbuf, atr);
            strcat(linbuf, " ");
            if(pos->siz==SIZNER) atr="NEAR";
            else if(pos->siz==SIZFAR) atr="FAR";
            else if(pos->siz==SIZBYT) atr="BYTE";
            else if(pos->siz==SIZWRD) atr="WORD";
            else if(pos->siz==SIZDWD) atr="DWORD";
            else if(pos->siz==SIZFWD) atr="FWORD";
            else if(pos->siz==SIZQWD) atr="QWORD";
            else atr="";
            strcat(linbuf, atr);
            lstputs(linbuf, 1);
            cnt++;
            pos++;
        }
    }
}
/* SYMBOLのSORT */
static void _near lblsrt(void) {
    int         qlen,
                xcnt,
                ycnt,
                xblk,
                yblk;
    struct lblt   *xpos,
                *ypos,
                *tpos,
                temp;
    /* SYMBOL TABLEを先頭に詰める */
    qlen=(int)sizeof(struct lblt);
    xpos=lbla[0];
    xblk=0;
    xcnt=0;
    while(1) {
        if(xcnt>=(int)lblb) {
            xblk++;
            if(xblk>=LBHSH) break;
            xpos=lbla[xblk];
            xcnt=0;
        }
        if((int)xpos->nam[0]=='¥0') {
            ycnt=xcnt+1;
            yblk=xblk;
            ypos=xpos+1;
            while(1) {
                if(ycnt>=(int)lblb) {
                    yblk++;
                    if(yblk>=LBHSH) break;
                    ypos=lbla[yblk];
                    ycnt=0;
                }
                if((int)ypos->nam[0]!='¥0') {
                    memcpy(xpos, ypos, qlen);
                    ypos->nam[0]=(char)'¥0';
                    break;
                }
                ycnt++;
                ypos++;
            }
            if((int)xpos->nam[0]=='¥0') break;
        }
        xcnt++;
        xpos++;
    }
    /* SORT */
    xpos=lbla[0];
    xblk=0;
    xcnt=0;
    while(1) {
        if(xcnt>=(int)lblb) {
            xblk++;
            if(xblk>=LBHSH) break;
            xpos=lbla[xblk];
            xcnt=0;
        }
        if((int)xpos->nam[0]=='¥0') break;
        tpos=xpos;
        yblk=xblk;
        ypos=xpos+1;
        ycnt=xcnt+1;
        while(1) {
            if(ycnt>=(int)lblb) {
                yblk++;
                if(yblk>=LBHSH) break;
                ypos=lbla[yblk];
                ycnt=0;
            }
            if((int)ypos->nam[0]=='¥0') break;
            if(strcmp(tpos->nam, ypos->nam)>0) tpos=ypos;
            ycnt++;
            ypos++;
        }
        if(tpos!=xpos) {
            memcpy(&temp, tpos, qlen);
            memcpy(tpos, xpos, qlen);
            memcpy(xpos, &temp, qlen);
        }
        xcnt++;
        xpos++;
    }
}
/* FILE PATHのPARSING */
static char * _near fnset(char *buf,char *nam) {
    int     ii;
    ii=64;
    if(*(buf+1)==(char)':') {
        *nam++=*buf++;
        *nam++=*buf++;
        ii-=2;
    }
    while(1) {
        if(*buf==(char)'.' && ii!=0) {
            *nam++=*buf++;
            break;
        }
```

アセンブラ用語集
条件分岐　フラグやCXレジスタの状況により分岐を決める．条件分岐は分岐先が128バイト以内という制限がある．

```
        if(ii==0 || strchr("/,:;[]<>¥"=+|?*",(int)*buf)!=NULL ||
            (int)*buf<=' ') {
            *nam=(char)'¥0';
            return buf;
        }
        *nam++=*buf++;
        ii--;
    }
    if(ii>3) ii=3;
    while(1) {
        if(ii==0 || strchr("/,.:;[]<>¥¥¥"=+|?*",(int)*buf)!=NULL
            || (int)*buf<=' ') {
            *nam=(char)'¥0';
            return buf;
        }
        *nam++=*buf++;
        ii--;
    }
    *nam=(char)'¥0';
    return buf;
}

/* 文字を繰り返して文字列を作成 */
static void _near strrpt(char *ptr, int chr, int num) {

    while(num>0) {
        *ptr++ =(char)chr;
        num--;
    }
    *ptr=(char)'¥0';
}

/* 入出力操作のERROR表示とABORT */
void asmerr(unsigned int code, char *admsg) {

    char    *msg;

    switch(code) {
    case 0:
        msg="メモリが足りません";
        break;
    case 1:
        msg="スイッチの指定が不正です";
        break;
    case 2:
        msg="余計なファイルの指定があります";
        break;
    case 3:
        msg="ファイル名の中に区切り文字があります";
        break;
    case 4:
        msg="ソースファイルの指定がありません";
        break;
    case 5:
        msg="ファイルが見つかりません";
        break;
    case 6:
        msg="ファイルを読込めません";
        break;
    case 7:
        msg="ファイルを作成できません";
        break;
    case 8:
        msg="ファイルへ書込めません";
        break;
    case 9:
        msg="ファイルをEOFまで読込ました";
        break;
    case 10:
        msg="ディスクが一杯です";
        break;
    case 11:
        msg="SEEK ERRORです";
    }
    if(code>=1 && code<=3) {
        printf("A%04X: %s", code, msg);
        if(admsg!=NULL) printf(" -- %s", admsg);
        printf("¥n");
    }
    else {
        printf("AF%03X: %s", code, msg);
        if(admsg!=NULL) printf(" -- %s", admsg);
        printf("¥n");
        exit(1);
    }
}

/* SOURCE PROGRAMの診断MESSAGEの表示 */
void errgen(unsigned int cod, char *admsg) {

    char    buf[258],
            *msg;

    if(passno!=3) return;
    switch(cod) {
    case 1:     /* LBL86 */
        msg="SYMBOLに使用できない文字があります";
        break;
    case 2:     /* LBL86 */
        msg="SYMBOLTABLEが一杯です";
        break;
    case 3:     /* LBL86 */
        msg="PHASE ERRORが発生しました";
        break;
    case 4:     /* LBL86 */
        msg="SYMBOLは多重に定義されています";
        break;
    case 5:     /* LBL86 */
        msg="未定義のSYMBOLを参照しています";
        break;
    case 6:     /* LBL86 */
        msg="多重定義のSYMBOLを参照しています";
        break;
    case 7:     /* EXP86 */
        msg="SHORT演算のOPERANDが不正です";
        break;
    case 8:     /* EXP86 */
        msg="AND/OR/XOR演算のOPERANDが不正です";
        break;
    case 9:     /* EXP86 */
        msg="NOT演算のOPERANDが不正です";
        break;
    case 10:    /* EXP86 */
        msg="比較演算のOPERANDが不正です";
        break;
    case 11:    /* EXP86 */
        msg="BASE REGISTERを多重に指定しています";
        break;
    case 12:    /* EXP86 */
        msg="INDEX REGISTERを多重に指定しています";
        break;
    case 13:    /* EXP86 */
        msg="SEGMENT OVERRIDEを多重に指定しています";
        break;
    case 14:    /* EXP86 */
        msg="加算演算子のOPERANDが不正です";
        break;
    case 15:    /* EXP86 */
        msg="減算演算のOPERANDが不正です";
        break;
    case 16:    /* EXP86 */
        msg="SCALEの値が不正です";
        break;
    case 17:    /* EXP86 */
        msg="乗算演算のOPERANDが不正です";
        break;
    case 18:    /* EXP86 */
        msg="除算・剰余・シフト演算のOPERANDが不正です";
        break;
    case 19:    /* EXP86 */
        msg="除算・剰余演算の除数が0です";
        break;
    case 20:    /* EXP86 */
        msg="-X演算のOPERANDが不正です";
        break;
    case 21:    /* EXP86 */
        msg="OFFSET演算のOPERANDが不正です";
        break;
    case 22:    /* EXP86 */
        msg="NEAR PTR演算のOPERANDが不正です";
        break;
    case 23:    /* EXP86 */
        msg="FAR PTR演算のOPERANDが不正です";
        break;
    case 24:    /* EXP86 */
        msg="BYTE/xWORD PTR演算のOPERANDが不正です";
        break;
    case 25:    /* EXP86 */
        msg="SEGMENT演算のOPERANDが不正です";
        break;
    case 26:    /* EXP86 */
        msg="要素あるいはOPERANDがありません";
        break;
    case 27:    /* EXP86 */
        msg="¥"(¥"に対応する¥")¥"がありません";
        break;
    case 28:    /* EXP86 */
        msg="¥"[]¥"の中に¥"[式]¥"は置けません";
        break;
    case 29:    /* EXP86 */
        msg="¥"[¥"に対応する¥"]¥"がありません";
        break;
    case 30:    /* EXP86 */
        msg="2進数の中に2進数以外の文字があります";
        break;
    case 31:    /* EXP86 */
        msg="16進数の中に16進数以外の文字があります";
        break;
    case 32:    /* EXP86 */
        msg="文字列が¥"¥'¥"で終わっていません";
        break;
    case 33:    /* EXP86 */
        msg="BASEまたはINDEX REGISTERが不正です";
        break;
    case 34:    /* EXP86 */
        msg="SCALEの対象がINDEX REGISTERでありません";
        break;
    case 35:    /* EXP86 */
        msg="OPERANDに余計な文字があります";
        break;
    case 36:    /* OP86 */
        msg="NEAR LABEL以外は使用できません";
        break;
    case 37:    /* OP86 */
        msg="無効な命令か、命令がありません";
        break;
    case 38:    /* EXP86 */
        msg="BASEおよびINDEXのREGISTERの組み合わせが不正です";
        break;
    case 39:    /* EXP86 */
        msg="INDEX REGISTER単独の使用はできません";
        break;
    case 40:    /* OP86 */
        msg="命令に対するOPERANDがありません";
        break;
    case 41:    /* PX86 */
        msg="CPUが186以上の命令です";
        break;
```


アセンブラ用語集
シンボル　アセンブラにおいて，ネームフィールドに記述する名前のこと．ラベルや変数名などは，すべてシンボルに含まれる．

```
        case 42:    /* PX86 */
            msg="CPUが286以上の命令です";
            break;
        case 43:    /* PX86 */
            msg="CPUが386以上の命令です";
            break;
        case 44:    /* PX86 */
            msg="CPUが486の命令です";
            break;
        case 45:    /* PX86 */
            msg="OPERANDのSIZEが不当です";
            break;
        case 46:    /* PX86 */
            msg="OPERANDのTYPEが不当です";
            break;
        case 47:    /* PX86 */
            msg="OPERANDの値が範囲を超えています";
            break;
        case 48:    /* OP86 */
            msg="SEGMENT OVERRIDEが不当です";
            break;
        case 49:    /* PX86 */
            msg="SEGMENT OVERRIDEはできません";
            break;
        case 50:    /* PX86 */
            msg="2つのOPERANDのサイズが異なります";
            break;
        case 51:    /* PX86 */
            msg="SYMBOLは使用できません";
            break;
        case 52:    /* PX86 */
            msg="PREFIX命令は使用できません";
            break;
        case 53:    /* PX86 */
            msg="定義するSYMBOLがありません";
            break;
        case 54:    /* PX86 */
            msg="NEAR LABELは使用できません";
            break;
        }
        sprintf(buf, "%s(%ld): AE%03X: %s.", srcfn, stncnt, cod, msg);
        if(admsg!=NULL) {
            strcat(buf, "--");
            strcat(buf, admsg);
        }
        if(linno==0) lsttl(0);
        lstputs(buf, 0);
        printf("%s¥n", buf);
        errcnt++;
}

/* SOURCE PROGRAMの致命的なERRORの表示とABORT */
void ftlgen(unsigned int cod) {

        char    *msg;

        switch(cod) {
        case 1:     /* PX86 */
            msg="OPERANDは前方参照できません";
            break;
        case 2:     /* PX86 */
            msg="OPERANDの式は許容できない式です";
            break;
        }
        cod+=0x0900;
        printf("%s(%ld): AF%03X: %s.--", srcfn, stncnt, cod, msg);
        printf("%s¥n", srcbuf);
        exit(3);
}

```

4-1 コマンドライン

コマンドラインの処理は，fioini関数の下請け関数cmdanlで行われる．簡易アセンブラのコマンドラインの書式は次の通りである．

AS86 ソースファイル名【オブジェクトファイル名
【リストファイル名】】【/L】【/N】【/O】【/Pn】【/Wn】

例によって【】は，【】で囲まれた要素が省略可能であることを示すもので，実際のコマンドラインに書いてはいけない．各要素は，1つ以上のスペース，タブまたはコンマ(,)で区切る必要がある．ただし，/L, /N, /O, /Pnおよび/Wnのスイッチは/が区切り文字の役目をかねているので，ファイル名や他のスイッチに続けて書いても構わない．また，スイッチの位置はソースファイルの前に置いてもよいし，ファイル名とファイル名の間にあってもよいが，ファイルの指定順序はソース，オブジェクト，リストの各ファイルの順でなければならない．

コマンドラインには，ソースファイル名は必ず書かなければならない．ソースファイル名は，アセンブルするソースプログラムのフルパス名で，あるいはカレントディレクトリの場合は，ファイルのベース名と拡張子で指定する．拡張子は省略可能であり，省略された場合は.ASMを拡張子として補う．

オブジェクトファイル名は，オブジェクトを出力するファイル名を指定する．オブジェクトファイル名は，指定任意である．もし指定されないときは，オブジェクトは/Oスイッチがない限り出力されない．拡張子が省略された場合は，.COMを拡張子として補う．

リストファイル名は，アセンブリリストを出力するファイル名を指定する．リストファイル名は，指定任意である．もし指定されないときは，/Lスイッチがない限りリストファイルは出力されない．また，リストファイルを指定するときは，オブジェクトファイルの指定も必ず行わなければならない．これは，cmdanl関数がファイル指定の出現順序によって，ファイルの用途を振り分けているためである．リストファイルの拡張子を指定しなかった場合は，.LSTを拡張子とする．

アセンブラ用語集
スタック　一時的にデータを格納する記憶領域を指し，サブルーチンや割り込み処理などの終了後に戻るアドレスやレジスタの内容を退避するのに使われる．

/Lスイッチは，リストファイルの指定がなかった場合，ソースファイル指定の拡張子を.LSTに変えてアセンブリリストを出力させる．リストファイルの指定を行った場合は，何も効果はない．このスイッチに対応する変数はlstsw変数である．ただし，実際にリストファイルを出力するかどうかは，lstfnのリストファイル名がヌル文字列かどうかで決定している．

/Nスイッチは，アセンブリリストの内，シンボルリストの出力を禁止するスイッチである．このスイッチを指定すると，lblsw変数をFALSEにする．この結果，シンボルリスト作成のための関数lbloutを呼び出しても何も実行しない．

/Oスイッチは，オブジェクトファイルの指定がない場合，ソースファイルの指定の拡張子の部分を.COMに変えてオブジェクトを出力させる．もし，オブジェクトファイルを指定していれば何の効果も発生しない．このスイッチに対応する変数は，objswである．ただし，実際にオブジェクトファイルを出力するかどうかは，アセンブリリストと同様にobjfnのオブジェクトファイル名がヌル文字列かどうかで決定している．

/Pnスイッチと/Wnスイッチは，アセンブリリストの書式を指定するものである．いずれのスイッチも，nの位置にはnの代わりに10進数を指定する．/Pnスイッチは，1ページの行数をnで指定する．nの指定可能な範囲は，30から256まででこの範囲外の値を指定すると66とみなされ，/Pnスイッチを指定しなかった場合も同様である．/Wnスイッチは，1行の文字数を指定する．nの範囲は，80から256である．nにこの範囲外の値を指定したり，/Wnスイッチの指定のないときはn＝132とみなされる．

たとえば，ソースファイルTEST.ASMをアセンブルしてオブジェクトファイル(.COM形式のロードモジュール)とアセンブリリストファイルTEST.LST作成するときは，

```
AS86 TEST /O /L
```

とMS-DOSのコマンドラインから入力すればよい．また，TEST.ASMをアセンブルしてオブジェクトは出力せず，アセンブリリストを標準出力装置に表示したいときは，

```
AS86 TEST NUL CON
```

のような指定もできる．付録のディスクには，簡易アセンブラAS86に対するバッチプログラムとして，TA86.BAT，OA86.BATおよびLA86.BATの3つのバッチファイルが格納されている．TA86は，オブジェクトファイルもリストファイルも出力しないで，単にエラーチェックだけを実行し，OA86.BATはオブジェクトファイルだけを作成し，LA86.BATはリストファイルのみを作成する．いずれのバッチプログラムも，オペランドには1つのソースプログラムファイルを記述する．たとえば，TEST.ASMのエラーチェックのみを実行するには，

```
TA86 TEST
```

のようにMS-DOSのコマンドラインから入力する．

4-2 ソースファイルのバッファリング

アセンブラでは，少しでも速度を向上させるためいろいろな工夫が施される．とくに，簡易アセンブラは3パス式であることと，C言語で記述していることもあって，残念ながらあまり高速性は望めない．しかし，できることはやっておこうという高速化の1つがこのソースファイルのバッファリングである．

3パス式アセンブラでは，同じソースファイルを3回読み込むことになる．もし，メモリの中に入れることができれば，1度メモリに格納してしまえば，後はディスク装置から読込む必要はなくなる．そこで，簡易アセンブラでは65280(0xFF00)バイトのバッファを確保して，パス1の開始時のsrcopn関数呼び出し時に，ソースプログラムを一気にこのバッファに読込むようにしている．そして，srcget関数が呼び出されると，このバッファの中から1行ずつsrcbufへ読み出す．read関数によってソースファイルから読み出されたバイト数が65280以下である場合には，ソースファイルはすべてバッファに読み込まれたことになる．このときは，srceof＝TRUE，srcbig＝FALSEとして，直ちにソースファイルをクローズする．そして，以降のsrcopn関数とsrccls関数ではファイルの操作は行わず，バッファからの読み出し位置を管理するポインタsrcpntとカウンタsrccntのみに操作を加

アセンブラ用語集
スタックセグメント　スタックとして格納されるデータのアドレスのセグメント．

えるようにする．

しかし，不幸にしてソースプログラムがバッファに入りきらないとき，すなわちread関数によってディスクから読み出されたバイト数が65280であった場合には，srceof = FALSE, srcbig = TRUEにしてソースがバッファに入りきらなかったことを示す．このときは，バッファの内容をすべてsrcbufに読み込むと，srcget関数の下請け関数であるsrcset関数の中でさらにファイルからソースプログラムを入力する．このときもソースの入力は，65280バイト単位で行う．そして，read関数によって読み出されたバイト数が65280バイトより少なくなると，srceof =TRUEにしてソースファイルがこれ以上読み込めないことを示す．srcbigはTRUEのままになっているので，このときはパスの最後にsrccls関数が呼び出されるとファイルをクローズする．

いずれの場合にしてもsrcget関数は，ソースプログラムが存在する以上に読み込もうとするとERRORを返す．このような事態になるのは，ソースプログラム中に正しいENDディレクティブがない場合である．簡易アセンブラでは，ソースファイルの最後にENDディレクティブがない場合，ENDディレクティブがあったものとして何も警告を出さないで処理を続けるようになっている．

4-3 オブジェクトファイルの形式

簡易アセンブラのオブジェクトは，MS-DOSの.COM形式のロードモジュールである．この形式では，アセンブリリストの機械コードがそのままファイルに書込まれる．ところで，COM形式のロードモジュールでは，オフセットアドレス0100Hの機械コードが先頭にこなければならない．しかし，常にこのような出力を行ったのでは汎用性がかなり失われてしまうことになる．たとえば，デバイスドライバの先頭アドレスは0000Hである．また，機械コードの配置アドレスを決めるためのディレクティブとしてORG命令があるが，最低位のアドレスから順番にORG命令を書いていくような制限を付けると，コーディングに不便な場合もある．そこで，簡易アセンブラではオブジェクトファイルの先頭におかれる機械コードは，コーディング順序によらず，ORGディレクティブで最低位アドレスに配置された機械コードがくるよう工夫している．

コーディング順序によらず，指定された位置に機械コードを出力するには，機械コードが配置された最低位アドレスをパス2で記憶しておけばよい．これを記憶しているのが，「3章」で述べたlacminである．ついでに，lacmaxには機械コードが配置された最高位+1番地のオフセットアドレスが記憶されている．このデータを使用すると，パス3で出力されるオブジェクトファイルのサイズはlacmax - lacminバイトとなる．そこで，objopn関数ではではオブジェクトファイルをオープンすると，このバイト数の0x00のデータをあらかじめオブジェクトファイルに出力している．このようにすると，ランダムな順序でORG命令が現れ，機械コードが出力されない領域の隙間があっても，常にその領域の隙間は0x00でクリアされていることになる．

ORGディレクティブが現れると，ファイルポインタを移動しなければならない．移動する位置は，ORGディレクティブのオペランドの値がxであったとすると，x-lacminである．これを行うのがobjorg関数で，ORG命令を処理する関数op63から呼び出される．たとえば，簡易アセンブラでは

```
        ORG     1000H
DATA1   DB      1
DATA2   DB      2
.............................
.............................
        ORG  0100H
START:  MOV  AX,DATA1
        MOV  BX,DATA2
.............................
.............................
        END
```

というような順序のプログラムをアセンブルしても，0100Hに配置される機械コードがファイルの先頭に配置されて，MS-DOSの.COMファイルとして正常に動作する．なお，ORGディレクティブがない場合には先頭の機械コードは0000Hから配置されるので.COMファイルとしてMS-DOSで実行することはできないので注意を要する．

4-4 アセンブリリスト

アセンブリリストに表示すべきものは，機械コードの配置アドレス，機械コードの内容，およびソースプログラ

アセンブラ用語集
ストリング　メモリブロック同様に連続するメモリ上のデータ列を指す．

ムの対応する行である。以前のアセンブラでは，ソースプログラムは，固定形式に変換して表示するのが普通であった。最近のアセンブラでは比較的長いシンボルが許され，字下げが行われたりするので，対応するソースプログラムの行の表示は，入力されたソースをそのまま表示するのが普通である。簡易アセンブラでも，srcbufの内容をそのまま表示している。

機械コードの配置アドレスを出力するためのデータは，locbufにmchopn関数とmchlst関数によって作成される。mchopn関数は，機械コードを生成する関数が呼び出される直前に呼び出され，アセンブリリストのlocbufの内容を作成すると同時に，機械コードの内容をリストに出力するためのバッファであるmchbufを初期化する。

機械コードを生成する関数は，1バイトの機械コードを出力するために，必ずmchbvst関数を呼び出す。この関数は，mchlst関数を呼び出してmchbufに16進数で機械コードの内容のデータを作成すると同時に，その機械コードをobjput関数でオブジェクトファイルにも出力するようになっている。2バイトあるいは4バイトの機械コードを出力するmchwvst関数とmchlvst関数も結局mchbvst関数を呼び出すので同様である。mchlst関数は，mchbufに機械コードが10バイト分蓄積されるとlstout関数を呼び出し，locbuf，mchbuf，stnbufおよびsrcbufの内容をリストファイルに出力する。なお，stnbufの内容は，srcget関数で作成される行番号のデータである。

mchlst関数は，この後locbufの内容を次の機械コードが配置されるアドレスにセットし，lstinhをTRUEにする。lstinhがTRUEにセットされていると，lstout関数は，stnbufとsrcbufの内容を出力しなくなる。そしてさらに機械コードが生成されると，配置アドレスと機械コードの内容だけを表示する。以上の動作はかなり複雑であるが，論より証拠，次のプログラムを書いてアセンブリリストをLA86.BATを使って出力させてみるとわかるはずである。

```
DB      '0123456789ABCDEF'
END
```

lstout関数は，locbuf，mchbuf，stnbufおよびsrcbufの内容を1つの文字列に編集して，lstputs関数に引き渡す。lstputs関数は，引き渡されたデータをページ管理と行の文字数の管理を行いながらリストファイルに出力する。

4-5 シンボルリスト

アセンブリリストの最後には，シンボルと値の対応表を付けるのが習慣である。最近では，シンボリックデバッガにファイルとして引き渡すことも行われているが，簡易アセンブラではリストに出力するだけである。

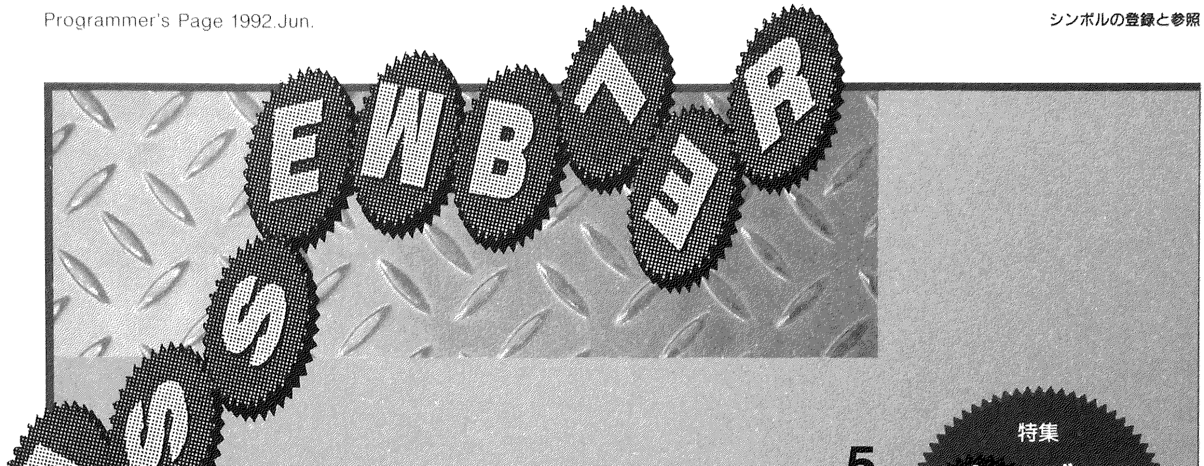
パス3の最後で，pass3関数がlblout関数を呼び出すとシンボルテーブルが作成される。lblout関数は，辞書順にシンボルを並べ変えるため，lblsrt関数を呼び出してシンボルのソートを行う。簡易アセンブラでは，シンボルをハッシングしているためソートは多少面倒で，時間のかかる作業を行わなければならない。lblsrt関数の前半では，ハッシングされてシンボルテーブルにまばらに登録されているシンボルをシンボルテーブルの先頭から順不同で詰め込む。そしてこの詰め込まれたシンボルを後半でバブルソートしている。あまり効率のよい方法ではないので，シンボルが多いと少し時間がかかるかもしれない。この辺の改良は，読者にぜひトライしていただきたい。

さて，ソートを終わったシンボルテーブルは，シンボルの名前，パス2での値および属性が1つのシンボルにつき1行のデータに編集されて，アセンブリリストで使用したのと同じ関数lstputsによりリストファイルに出力される。

4-6 エラー処理

プログラム3には，3つのエラー処理のための関数asmerr，errgenおよびfltgenが含まれている。これが簡易アセンブラのすべてのエラー処理である。asmerrは，メモリ確保，コマンドラインの解析およびファイルの操作を行う際発生するエラーの処理である。errgenは，機械コードの生成の際ソースプログラムに不法な記述があった場合に呼び出される。fltgenは，パス1のIFとORG命令のオペランドに処理できない記述があった場合に呼び出される。これらの関数は比較的簡単な処理であるから，詳細はプログラムを参照していただきたい。

アセンブラ用語集
セグメント　8086のアドレス空間は1MBある。このアドレスを指定する方法としてSegmentという概念を導入した。



特集 アセンブラの作り方

5 シンボルの登録と参照

アセンブラの最大のメリットは，シンボルやニモニックといった比較的覚えやすい略語や単語が使用できることである．このためアセンブラは，シンボルやニモニックなどをテーブルとして保持し，そのテーブルを参照することによってアセンブリ言語を解釈して，機械コードを生成する．プログラム4(LBL86.C)は，シンボルテーブルにシンボルを登録したり，登録されているシンボルを参照して，シンボルに与えられた値を求めたりするのに必要な関数群である．

5-1 シンボルのハッシング

アセンブラで最も時間のかかる仕事は，テーブルの検索である．そこで，テーブルの検索を能率的に行うことは，極めて重要なことである．テーブルを検索する最も単純な方法は，先頭から順にすべてのデータを登録しておいて，それを先頭から順に検索する方法である．しかし，この方法では登録したデータが多くなると，最後の方にあるデータを検索するには，かなり時間がかかることになる．よく知られた効率的な検索方法には，２分検索法がある．しかし，この方法ではデータが昇順あるいは降順にソートされていなければならず，ランダムに発生するシンボルの登録には不向きである．

簡易アセンブラで採用した方法は，ハッシング法と呼ばれる方法である．この方法では，シンボルテーブルをいくつかのブロックに分割して，登録されるシンボルを一定の基準で分割したブロックに分散させて登録する．このようにブロックに分散させると，通常は１つのブロック内だけ検索を行えばよいので，検索のための文字列の比較回数が減少し，検索時間が短縮されるのである．

シンボルを分散させるにはいろいろな方法があるが，もちろん単純な方法が時間的に有利なのはいうまでもない．そこで最も単純な方法として知られているトータルハッシュ法を採用する．この方法では，シンボルの１文字１文字の文字コードを数値とみなしてすべて加算する．そして，この数値がブロック数に収まる数値になるよう細工するのである．ここでは，この細工の方法としてブロック数で除算して余りを求め，それを登録するブロックの番号としている．たとえば，ABCというシンボルがあり，ブロック数はヘッダのLBHSHにあるように64であるとすると，Aの文字コードは10進数で64，Bは65，Cは66であるから，

ABCのブロック番号=(64+65+66) % 64
=6

と計算でき，第０ブロックから始まる第６ブロックに登録すべきシンボルであることがわかる．

実際にこの方法を適用すると問題が発生することがある．それは，偶然に１つのブロックにシンボルが集中する場合が希にあるからである．同じブロック番号を持つシンボルのことをシノニムと呼ぶが，シノニムが多く発生すると１つのブロックに登録できなくなってしまう．このような場合には簡易

アセンブラ用語集
セグメントアドレス　アドレスを指定したとき，この値を４ビット左にシフトしてオフセットアドレスを加えて実アドレスを決める．

リスト 4　LBL86.C

```
/****************************************/
/*  SYMBOLの処理(AS86)                  */
/****************************************/

#include <STDIO.H>
#include <STRING.H>
#include "AS86.H"

/* 予約語のTABLE */
static  char
    rsvwrd[70][8] = {
        "AL",     "CL",     "DL",     "BL",
        "AH",     "CH",     "DH",     "BH",
        "AX",     "CX",     "DX",     "BX",
        "SP",     "BP",     "SI",     "DI",
        "EAX",    "ECX",    "EDX",    "EBX",
        "ESP",    "EBP",    "ESI",    "EDI",
        "CS",     "ES",     "SS",     "DS",
        "FS",     "GS",     "CR0",    "CR2",
        "CR3",    "DR0",    "DR1",    "DR2",
        "DR3",    "DR6",    "DR7",    "TR3",
        "TR4",    "TR5",    "TR6",    "TR7",
        "MOD",    "SHL",    "SHR",    "NOT",
        "AND",    "OR",     "XOR",    "GT",
        "GE",     "EQ",     "NE",     "LE",
        "LT",     "BYTE",   "WORD",   "DWORD",
        "FWORD",  "QWORD",  "FAR",    "NEAR",
        "PTR",    "OFFSET", "SHORT",  "DUP",
        "$",      "?"     };

static int _near lblhash(char *lb);
static unsigned long _near rembyt(void);
static void _far * _near ptradd(void _far * ba, unsigned long by);

/* SYMBOL TABLEの初期化 */
int lblini(void) {
    int             ii;
    unsigned long llen,
                    remain;
    struct lblt     *ptr;

    llen=(unsigned long)sizeof(struct lblt);
    remain=rembyt();
    if(remain<=0x8000L) asmerr(0, "SYMBOL登録領域");
    remain-=0x8000L;
    if(remain<=(unsigned long)(LBHSH*sizeof(struct lblt)))
        asmerr(0, "SYMBOL登録領域");
    lbla[0]=(struct lblt *)alloc(remain);
    if(lbla[0]==NULL) asmerr(0, "SYMBOL登録領域");
    lbls=remain/LBHSH;
    lblb=lbls/llen;                     /* 登録可能数/ENTRY */
    lbls=lblb*llen;                     /* 端数切り捨て */
    for(ii=1; ii<LBHSH; ii++)   /* ENTRY毎の先頭ADDRESS */
        lbla[ii]=(struct lblt _far *)ptradd(lbla[ii-1], lbls);
    for(ii=0; ii<LBHSH; ii++) {     /* SYMBOL登録領域の初期化 */
        ptr=lbla[ii];
        memset(ptr, 0, (unsigned int)lbls);
    }
    return NORMAL;
}

/* SYMBOLの登録 */
void lblreg(unsigned long vl, unsigned int at, unsigned int sz) {

    struct lblt   *pos;

    at&=SPC;
    if((int)lblbuf[0]=='¥0') return;/* SYMBOLなし */
    if(lblchk(lblbuf)) {            /* 綴が不正 */
        errgen(1, lblbuf);
        return;
    }
    pos=lblref(lblbuf);
    if(pos==NULL) {                 /* 登録余地なし */
        if(passno==3) errgen(2, lblbuf);
        return;
    }
    if((int)pos->nam[0]=='¥0') {    /* 未登録のSYMBOL */
        memset(pos, 0, sizeof(struct lblt));
        strcpy(pos->nam, lblbuf);
    }
    if(passno==1) {                 /* PASS1の値の登録 */
        if((pos->att & DEF)) pos->att |= MLT;
        pos->att = (pos->att & ~SPC)|(at|DEF);
        pos->siz=sz;
        pos->p1val=vl;
        pos->p2val=vl;
    }
    if(passno==2) {                 /* PASS2の値の登録 */
        pos->att=(pos->att & ~SPC)|(at|DEF);
        pos->siz=sz;
        pos->p2val=vl;
    }
    if(passno==3) {                 /* PHASE ERRORのCHECK */
        if((pos->att & SPC)!=at) {
            printf("ATR %04X %04X¥n", pos->att&SPC, at);
            errgen(3, lblbuf);
        }
        else if(pos->siz != sz) {
            printf("SIZ %04X %04X¥n", pos->siz, sz);
            errgen(3, lblbuf);
        }
        else if(pos->p2val != vl) {
            printf("VAR %081X %081X¥n", pos->p2val, vl);
            errgen(3, lblbuf);
        }
        if((pos->att & MLT)!=0) errgen(4, lblbuf);
    }
}

/* SYMBOLの参照(式の要素) */
void lblev(char *pt) {

    struct lblt     *pos;

    memset(&expval, 0, sizeof(struct opdv));
    pos=lblref(pt);
    if(pos==NULL) {                     /* SYMBOL TABLEが一杯 */
        errgen(2, pt);
        return;
    }
    if((int)pos->nam[0] == '¥0') {  /* 未登録のSYMBOLの参照 */
        errgen(5, pt);
        if(lblchk(pt)) {
            memset(pos, 0, sizeof(struct lblt));
            strcpy(pos->nam, pt);
            pos->att=REF;
        }
        return;
    }
    if((pos->att & DEF)==0) {   /* 未定義のSYMBOLの参照 */
        errgen(5, pt);
        return;
    }
    if(pos->att & MLT) {                /* 多重定義のSYMBOLの参照 */
        errgen(6, pt);
        return;
    }
    pos->att |= REF;
    expval.att = (expval.att & SPC)|DEF;
}

/* SYMBOLの登録位置を見つける */
struct lblt *lblref(char *lb) {

    int             sum,
                    ii,
                    jj;
    struct lblt     *ptr;

    memset(&expval, 0, sizeof(struct opdv));
    sum=lblhash(lb);
    ptr=lbla[sum];
    for(jj=0; jj<(int)lblb; jj++) {     /* HOME ENTRYを探す */
        if((int)ptr->nam[0]=='¥0') goto L1;
        if(strcmp(lb, ptr->nam)==0) goto L2;
        ptr++;
    }
    for(ii=sum+1; ii<LBHSH; ii++) {     /* 前方ENTRYを探す */
        ptr=lbla[ii];
        for(jj=0; jj<(int)lblb; jj++) {
            if((int)ptr->nam[0]=='¥0') goto L1;
            if(strcmp(lb, ptr->nam)==0) goto L2;
            ptr++;
        }
    }
    for(ii=0; ii<sum; ii++) {       /* 後方ENTRYを探す */
        ptr=lbla[ii];
        for(jj=0; jj<(int)lblb; jj++) {
            if((int)ptr->nam[0]=='¥0') goto L1;
            if(strcmp(lb, ptr->nam)==0) goto L2;
            ptr++;
        }
    }
    /* 登録する余地無し */
    expval.att = NUM;
    return NULL;
L1:   /* 未登録 */
    expval.att = NUM;
    return ptr;           /* 登録すべき位置 */
L2:   /* 登録済み */
    expval.att = (ptr->att & DEF) | (ptr->att & SPC);
    expval.siz = ptr->siz;
    expval.p1val = ptr->p1val;
    expval.p2val = ptr->p2val;
    return ptr;           /* 登録されている位置 */
}

/* SOURCE PROGRAMをTOKENに分解する */
char *token(char *pt) {

    char    *dst;
    int     chr,
            cnt;

    dst=tknbuf;
    cnt=0;
    pt=pt+strspn(pt, "¥t ");            /* 空白の読み飛ばし */
    chr=(int)*pt;
    if(chr=='¥'') {                     /* 文字列 */
        *dst++ = *pt++;
        cnt=1;
        while(1) {
            chr=(int)*pt;
            if(chr=='¥0') break;
            if(chr=='¥'') {
                *dst++ =(char)chr;
                pt++;
                chr=(int)*pt;
                if(chr!='¥'') break;
            }
            else *dst++ = (char)chr;
```

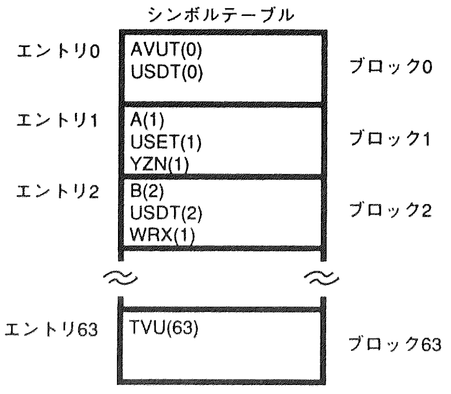

アセンブラ用語集
セグメントレジスタ　コード領域, スタック領域やデータ領域のセグメントアドレスを格納するレジスタ.

```
            pt++;
        }
    }
    /* 区切り文字・演算子 */
    else if(strchr("()*+,-/:;[]", chr)!=NULL) {
        if(chr!='¥0') *dst++ = *pt++;
    }
    /* SYMBOL・REGISTER・一部の演算子 */
    else {
        while(1) {
            chr=(int)*pt;
            if(strchr("¥t ()*+,-/:;[]", chr)!=NULL) break;
            if(cnt<LBLMAX) {
                *dst++ =(char)chr;
                cnt++;
            }
            pt++;
        }
    }
    *dst=(char)'¥0';
    return pt;
}

/* SYMBOLのHASHING */
static int _near lblhash(char *lb) {

    unsigned int    sum,
            chr;

    sum=0;
    while(1) {
        chr=(unsigned int)*lb++;
        if(chr==0) break;
        sum+=chr;
    }
    return sum%LBHSH;
}

/* SYMBOLに含まれる文字のCHECK */
int lblchk(char *lb) {

    char    *wk;
    int     ii,
            chr;

    chr=(int)*lb;
    if(chr=='¥0') return ERROR;
    if(strchr("0123456789", chr)!=NULL) return ERROR;
    wk=rsvwrd[0];
    for(ii=0; ii<70; ii++, wk+=8) /* [70][8] */
        if(stricmp(wk, lb)==0) return ERROR;
    return NORMAL;
}

/* MEMORYの使用可能量を求める */
#pragma optimize("le", off)
static unsigned long _near rembyt(void) {

    unsigned long by;

    _asm {
        MOV   AH,48H
        MOV   BX,0FFFFH
        INT   21H
        MOV   AX,BX
        XOR   DX,DX
        ADD   AX,AX
        ADC   DX,DX
        ADD   AX,AX
        ADC   DX,DX
        ADD   AX,AX
        ADC   DX,DX
        ADD   AX,AX
        ADC   DX,DX
        MOV   WORD PTR by,AX
        MOV   WORD PTR by[2],DX
```

アセンブラでは，多少検索効率は落ちるが，次のブロック以降の空き領域にあふれたシンボルを登録することでこの問題を解決している．図5は，簡易アセンブラで採用しているシンボルテーブルの様子を示したものであるが，シンボルWRXでこのような状況が発生している．

このような方法に基づき，lblbufに格納されたシンボルのシンボルテーブルに登録すべき位置あるいは登録されている位置を見つけるのがlblref関数である．lblref関数では，lblhash関数を呼び出すが，この関数がブロックの番号を計算する．シンボルを登録すべきブロックをホームブロックというが，lblref関数はホームブロックを探してもシンボルを発見できず，ホームブロックが一杯の場合は次のブロックを順次検索するようになっている．

図5　ハッシングされたシンボルテーブル



注：（）内の数字は文字コードを加算して64で割った余り

5-2 シンボルテーブルの領域確保と初期化

シンボルテーブルの領域確保と初期化は，main関数から起動時に1回lblini関数を呼び出すことで実行される．lblini関数では，システムの保有するメモリの残量をインラインアセンブラで記述したrembyt関数を呼び出して確認し，32KBを残してすべてのメモリをalloc関数で確保する．そして，そのメモリをLBHSH(64)個のブロックに配分する．各ブロックの先頭アドレスは，これもインラインアセンブラで記述したptradd関数で計算され，lblaにポインタの配列として格納される．

これ以降，シンボルテーブルの空き領域の判定は，シンボルの名前の先頭の文字が'¥0'かどうかで行うので，シ

アセンブラ用語集
ゼロフラグ　演算の結果がゼロの時セットされるフラグ．

```
    }
    return by;
}

/* MEMORYを割り当てる */
void _far * alloc(unsigned long by) {

    void    _far * ba;

    _asm {
        MOV   AX,WORD PTR by
        MOV   DX,WORD PTR by[2]
        ADD   AX,15
        ADC   DX,0
        SHR   DX,1
        RCR   AX,1
        SHR   DX,1
        RCR   AX,1
        SHR   DX,1
        RCR   AX,1
        SHR   DX,1
        RCR   AX,1
        MOV   BX,AX
        MOV   AH,48H
        INT   21H
        JNC   L1
        XOR   AX,AX
        MOV   DX,AX
        JMP   SHORT L2
    L1:
        MOV   DX,AX
        XOR   AX,AX
    L2:
        MOV   WORD PTR ba,AX
        MOV   WORD PTR ba[2],DX
    }
    return ba;
}

/* alloc関数で割り当てたメモリを解放する */
void mfree(void _far * ba) {

    _asm {
        MOV   ES,WORD PTR ba[2]
        MOV   AH,49H
        INT   21H
    }
}

/* FAR POINTERとLONG整数の加算 */
static void _far * _near ptradd(void _far * ba, unsigned long by) {

    _asm {
        MOV   AX,WORD PTR by
        MOV   DX,WORD PTR by[2]
        ADD   AX,WORD PTR ba
        ADC   DX,0
        MOV   BX,AX
        AND   AX,0FH
        MOV   WORD PTR ba,AX
        MOV   AX,BX
        SHR   DX,1
        RCR   AX,1
        SHR   DX,1
        RCR   AX,1
        SHR   DX,1
        RCR   AX,1
        SHR   DX,1
        RCR   AX,1
        ADD   WORD PTR ba[2],AX
    }
    return ba;
}
#pragma optimize("le", on)

```

ンボルテーブルはmemset関数で0クリアしている．先頭の文字が'¥0'であれば，その領域は未使用であると判定される．

5-3 シンボルの登録と参照

シンボルの登録が行われる場合は2つある．1つはlblev関数によってシンボルが参照されたとき未登録の場合である．この場合にシンボルを登録するのは，もし未定義のシンボルがあった場合，シンボルをシンボルリストに未定義としてリストアップするためである．参照時に登録される場合は，シンボルは必ず未定義であるからシンボルの属性(att)のDEFビットを0として登録する．また，シンボルが参照されたという意味でREFビットを1にセットしているが，このビットは簡易アセンブラでは使用していない．

もう1つの場合は，ソースプログラムの構文解析の結果，シンボルありと判定された場合である．この場合は，lblreg関数を使用して値と属性を指定して登録される．パス1では，これはシンボルの正式な登録であるので，DEFビットが1にセットされる．前記のlblev関数で未定義のシンボルとして登録されている場合には，登録されている位置の属性のDEFビットを1にして，関数の引数として指定されている属性と値を登録するのみである．また，すでにDEFビットが1であるシンボルにパス1でさらに登録を行おうとすると，属性のMLTビットを1にセットして二重以上にシンボルが登録された状態を保存している．MLTビットへの参照が行われると，そのたびにエラーを発生させるためである．シンボルの値の登録は，パス1では処理の都合上パス1の値(p1val)とパス2の値(p2val)の両方に行い，パス2では，パス2の値にのみ登録を行っている．パス3では，パス2で登録された値とパス3の値の比較を行う．これは，パス2とパス3の値が同じでなければ，機械コードのつじつまが合わないことになり，正常に実行できないからである．パス2とパス3でシンボルの値が合わなくなることをフェーズエラーと呼んでいる．バージョン5.1以前のMASMの使用の経験のある読者は，このエラーを体験した方が多いと思う．

シンボルの参照は，オペランドの値を求めるときexpr関数からlblev関数が呼び出されて行われる．シンボルが未定義のとき，あるいは前述のMLTビットが1であるときにはエラーが発生する．シンボルの属性と値は，lblt型構造体からexpvalのopdv型構造体の同じ名称のメンバにコピーされる．

5-5 シンボルに使用できない文字と語

アセンブラ用語集
ソース　ディスティネーションに対して処理の基となるオペランドを指す．

ソースプログラムの構文解析を行うときやオペランドの値を求めるとき，命令や演算子，レジスタ記号，区切り記号などと同じ文字や語は混乱の元となるので使用を禁止している．このチェックは，lblchk関数とソースプログラムの構文解析を行うsrclin関数で行われる．以下のa～dは，シンボルに使用できない文字と語である．

a．ラベルの先頭の文字として数字の0～9

b．ラベルのすべての文字として" ", " *", "+", " -", "/", " (", ")", ",", " ;", " [" および"]" の" で囲まれた文字．

c．プロセッサ命令とディレクティブのニモニック

d．プログラム4のrsvwrdにあがっているレジスタ名や演算子など

シンボル自身の間では，英大文字小文字の区別をつけているが，上記cとdについては大文字と小文字の区別をしていないので，大文字と小文字の区別なく同じスペルになるシンボルは使用できない．

特集 アセンブラの作り方

6 機械コードの生成

機械コードの生成は，簡易アセンブラのプログラムの量のかなりの部分を占めるが，シンボルの登録と参照および次の「7章」で述べる式の部分ができあがれば，どちらかというと力仕事に近い部分である．もちろん，ここで間違った機械コードを作成するようなことがあっては致命的である．プログラム5（OP86.C）は，命令テーブル，機械コード生成の入り口である構文解析および機械コードを生成する関数から共通に利用される関数をまとめたものである．

A 6-1 命令テーブル

命令テーブルは，構文解析によってソースプログラムから命令のニモニックを発見し，機械コードを生成するためのニモニックと機械コードの対応表のようなものである．プログラム5の最初にあるoptbl構造体が，命令テーブルの書式を規定している．命令テーブル本体は次のoplstである．命令テーブルは，シンボルテーブルと同様に，31のブロックにハッシングされている．

各命令テーブルの要素は，プロセッサ命令とディレクティブのニモニック，命令のグループ番号およびグループ内で，特定の命令を識別するための必要最小限の機械コードである．ハッシングブロックの終わりは，命令のニモニックがヌル文字列であることで行っている．シンボルテーブルと異なり，ホームブロック以外の場所に登録されている命令はない．

命令のグループ分けは，次のような分類を経て行っている．

リスト5 P86.C

```
/******************************************/
/*  構文解析・OBJECT生成(AS86)         */
/******************************************/

#include <STDLIB.H>
#include <STDIO.H>
#include <STRING.H>
#include "AS86.H"

#define OPMAX 7                 /* MNEMONICの文字長 */
#define OPHSH 31                /* HASHING ENTRY数 */
#define OPENT 16                /* 1BLOCKの最大命令数 */

struct optbl {                  /* 命令TABLEの形式 */
    char    name[OPMAX+1];          /* MNEMONIC */
    int     group,                  /* 所属するGROUP */
            code;               /* 命令識別CODE */
};

/* 命令TABLE */
static struct optbl oplst[OPHSH][OPENT] = {
    {{"BTC",    44, 0xBB38}, {"CLTS",    4, 0x0F06},     /* [0] */
     {"DW",     57, 0x0000}, {"JNA",    11, 0x0076},
     {"LODSB",   1, 0x00AC}, {"OUT",    33, 0x00E6},
     {"POPAD",   6, 0x6661}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000}},
    {{"CMPSB",   1, 0x00A6}, {"IMUL",   48, 0x0028},     /* [1] */
     {"INC",     8, 0x0040}, {"JNB",    11, 0x0073},
     {"LOOPZ",  16, 0x00E1}, {"NEG",     9, 0x0018},
     {"SETNZ",  23, 0x0F95}, {"SLDT",   22, 0x0000},
     {"STR",    22, 0x0008}, {"XOR",    26, 0x0030},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000}},
    {{"BSF",    43, 0x0FBC}, {"ESC",    28, 0x00D8},     /* [2] */
     {"JLE",    11, 0x007E}, {"JNC",    11, 0x0073},
     {"JS",     11, 0x0078}, {"LODSD",   6, 0x66AD},
     {"SETL",   23, 0x0F9C}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000}},
    {{"CBW",     1, 0x0098}, {"CMPSD",   6, 0x66A7},     /* [3] */
     {"XLAT",   20, 0x00D7}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000}},
    {{"BOUND",  37, 0x0062}, {"INTO",    1, 0x00CE},     /* [4] */
     {"IRETD",   6, 0x66CF}, {"JNE",    11, 0x0075},
     {"LAHF",    1, 0x009F}, {"LOOP",   16, 0x00E2},
     {"SETGE",  23, 0x0F9D}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000}},
    {{"CALL",    7, 0x0000}, {"CWD",     1, 0x0099},     /* [5] */
     {"POPFD",   6, 0x669D}, {"SETO",   23, 0x0F90},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000}},
    {{"JNG",    11, 0x007E}, {"JPE",    11, 0x007A},     /* [6] */
     {"LAR",    40, 0x0F02}, {"OR",     26, 0x0008},
     {"REPNE",   2, 0x00F2}, {"SETP",   23, 0x0F9A},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000}},
    {{"CMP",    25, 0x0038}, {"JNAE",   11, 0x0072},     /* [7] */
     {"SAL",    34, 0x0020}, {"SETNA",  23, 0x0F96},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000}},
    {{"ASIZE",   5, 0x0067}, {"JNBE",   11, 0x0077},     /* [8] */
     {"RCL",    34, 0x0010}, {"SETNB",  23, 0x0F93},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000}},
    {{"AAA",     1, 0x0037}, {"BSWAP",  24, 0x0FC8},     /* [9] */
     {"JCXZ",   16, 0x00E3}, {"JZ",     11, 0x0074},
     {"MOVSW",   1, 0x00A5}, {"SETLE",  23, 0x0F9E},
     {"SETNC",  23, 0x0F93}, {"SETS",   23, 0x0F98},
     {"VERR",   22, 0x0020}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000}},
    {{"DB",     55, 0x0000}, {"DIV",     9, 0x0030},     /* [10] */
     {"ENTER",  38, 0x00C8}, {"LDS",    30, 0x00C5},
     {"MOVSX",  46, 0x0FBE}, {"PUSH",   18, 0x0000},
     {"TEST",   35, 0x0000}, {"XADD",   47, 0x0FC0},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000}},
    {{"INSW",    3, 0x006D}, {"JNL",    11, 0x007D},     /* [11] */
     {"LES",    30, 0x00C4}, {"LOOPE",  16, 0x00E1},
     {"REPZ",    2, 0x00F3}, {"SAHF",    1, 0x009E},
     {"SETNE",  23, 0x0F95}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000}},
    {{".8086",  54, 0x0000}, {"AAD",     1, 0xD50A},     /* [12] */
     {"CWDE",    6, 0x6698}, {"DAA",     1, 0x0027},
     {"LFS",    45, 0x0FB4}, {"DD",     56, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000}},
    {{"JNGE",   11, 0x007C}, {"LGS",    45, 0x0FB5},     /* [13] */
     {"LMSW",   22, 0x0130}, {"PUSHA",   3, 0x0060},
     {"SAR",    34, 0x0038}, {"SCASW",   1, 0x00AF},
     {"SETNG",  23, 0x0F9E}, {"SETPE",  23, 0x0F9A},
     {"STOSW",   1, 0x00AB}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000}},
    {{"ADC",    25, 0x0010}, {"BSR",    43, 0x0FBD},     /* [14] */
     {"JMP",    13, 0x0000}, {"JNO",    11, 0x0071},
     {"RCR",    34, 0x0018}, {"REP",     2, 0x00F3},
     {"SETNAE", 23, 0x0F92}, {"SHL",    34, 0x0020},
     {"VERW",   22, 0x0028}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000}},
    {{"ADD",    25, 0x0000}, {"BTR",    44, 0xB330},     /* [15] */
     {"HLT",     1, 0x00F4}, {"JA",     11, 0x0077},
     {"JNP",    11, 0x007B}, {"MOVS",   27, 0x00A4},
     {"ORG",    63, 0x0000}, {"OUTSW",   3, 0x006F},
     {"SETNBE", 23, 0x0F97}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000}},
    {{"BTS",    44, 0xAB28}, {"JB",     11, 0x0072},     /* [16] */
     {"JECXZ",  12, 0x67E3}, {"JPO",    11, 0x007B},
     {"SETZ",   23, 0x0F94}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000}},
    {{"INS",    39, 0x006C}, {"JC",     11, 0x0072},     /* [17] */
     {"LOOPNZ", 16, 0x00E0}, {"MOVZX",  46, 0x0FB6},
     {"STC",     1, 0x00F9}, {"SUB",    25, 0x0028},
     {"ENDIF",  60, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000}},
    {{"DEC",     8, 0x0048}, {"EQU",    61, 0x0000},     /* [18] */
     {"INT",    10, 0x00CD}, {"JNLE",   11, 0x007F},
     {"JNS",    11, 0x0079}, {"LOCK",    2, 0x00F0},
     {"LSL",    40, 0x0F03}, {"PUSHF",   1, 0x009C},
     {"RET",    19, 0x00C2}, {"SETNL",  23, 0x0F9D},
     {"STD",     1, 0x00FD}, {"ELSE",   58, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000}},
    {{".186",   50, 0x0000}, {"JE",     11, 0x0074},     /* [19] */
     {"MOVSB",   1, 0x00A4}, {"PUSHAD",  6, 0x6660},
     {"SCAS",   15, 0x00AE}, {"STOS",   15, 0x00AA},
     {"XCHG",   36, 0x0086}, {"IF",     62, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000}},
    {{".286",   51, 0x0000}, {"LGDT",   21, 0x0110},     /* [20] */
     {"NOP",     1, 0x0090}, {"ROL",    34, 0x0000},
     {"SETNGE", 23, 0x0F9C}, {"SHLD",   49, 0x0FA4},
     {"SHR",    34, 0x0028}, {"SMSW",   22, 0x0120},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000}},
    {{".386",   52, 0x0000}, {"AAM",     1, 0xD40A},     /* [21] */
     {"IDIV",    9, 0x0038}, {"INSB",    3, 0x006C},
     {"JG",     11, 0x007F}, {"LODSW",   1, 0x00AD},
     {"MOVSD",   6, 0x66A5}, {"MUL",     9, 0x0020},
     {"OUTS",   41, 0x006E}, {"REPE",    2, 0x00F3},
     {"SETNO",  23, 0x0F91}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000}},
    {{".486",   53, 0x0000}, {"CMPSW",   1, 0x00A7},     /* [22] */
     {"JAE",    11, 0x0073}, {"LIDT",   21, 0x0118},
     {"OSIZE",   5, 0x0066}, {"POP",    17, 0x0000},
     {"SETA",   23, 0x0F97}, {"SETNP",  23, 0x0F9B},
     {"",        0, 0x0000}, {"",        0, 0x0000,
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000}},
    {{"INSD",    6, 0x666D}, {"JBE",    11, 0x0076},     /* [23] */
     {"SCASB",   1, 0x00AE}, {"SETB",   23, 0x0F92},
     {"SETPO",  23, 0x0F9B}, {"STI",     1, 0x00FB},
     {"STOSB",   1, 0x00AA}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000},
     {"",        0, 0x0000}, {"",        0, 0x0000}},
    {{"ARPL",   42, 0x0063}, {"CLC",     1, 0x00F8},     /* [24] */
```

アセンブラ用語集
ダブルワード　32ビット（4バイト）で表せる値をワードといい，単精度浮動小数点などを表すのに使われる．

```c
        {"LEA",     31, 0x008D}, {"LEAVE",    3, 0x00C9},
        {"NOT",      9, 0x0010}, {"PUSHFD",   6, 0x669C},
        {"SETC",    23, 0x0F92}, {"WBINVD",   4, 0x0F09},
        {"",         0, 0x0000}, {"",         0, 0x0000},
        {"",         0, 0x0000}, {"",         0, 0x0000},
        {"",         0, 0x0000}, {"",         0, 0x0000},
        {"",         0, 0x0000}, {"",         0, 0x0000}},
       {{"AND",     26, 0x0020}, {"CLD",      1, 0x00FC},     /* [25] */
        {"CMC",      1, 0x00F5}, {"JNZ",     11, 0x0075},
        {"LLDT",    22, 0x0010}, {"LSS",     45, 0x0FB2},
        {"LTR",     22, 0x0018}, {"MOV",     32, 0x0000},
        {"OUTSB",    3, 0x006E}, {"POPA",     3, 0x0061},
        {"SCASD",    6, 0x66AF}, {"SETNLE",  23, 0x0F9F},
        {"SETNS",   23, 0x0F99}, {"STOSD",    6, 0x66AB},
        {"",         0, 0x0000}, {"",         0, 0x0000}},
       {{"BT",      44, 0xA320}, {"CMPXCHG", 47, 0x0FB0},     /* [26] */
        {"JL",      11, 0x007C}, {"RETF",    19, 0x00CA},
        {"ROR",     34, 0x0008}, {"SETE",    23, 0x0F94},
        {"SHRD",    49, 0x0FAC}, {"INVD",     4, 0x0F08},
        {"",         0, 0x0000}, {"",         0, 0x0000},
        {"",         0, 0x0000}, {"",         0, 0x0000},
        {"",         0, 0x0000}, {"",         0, 0x0000},
        {"",         0, 0x0000}, {"",         0, 0x0000}},
       {{"AAS",      1, 0x003F}, {"IN",      29, 0x00E4},     /* [27] */
        {"LODS",    14, 0x00AC}, {"LOOPNE",  16, 0x00E0},
        {"OUTSD",    6, 0x666F}, {"REPNZ",    2, 0x00F2},
        {"SGDT",    21, 0x0100}, {"",         0, 0x0000},
        {"",         0, 0x0000}, {"",         0, 0x0000},
        {"",         0, 0x0000}, {"",         0, 0x0000},
        {"",         0, 0x0000}, {"",         0, 0x0000},
        {"",         0, 0x0000}, {"",         0, 0x0000}},
       {{"CMPS",    27, 0x00A6}, {"JGE",     11, 0x007D},     /* [28] */
        {"SETG",    23, 0x0F9F}, {"",         0, 0x0000},
        {"",         0, 0x0000}, {"",         0, 0x0000},
        {"",         0, 0x0000}, {"",         0, 0x0000},
        {"",         0, 0x0000}, {"",         0, 0x0000},
        {"",         0, 0x0000}, {"",         0, 0x0000},
        {"",         0, 0x0000}, {"",         0, 0x0000},
        {"",         0, 0x0000}, {"",         0, 0x0000}},
       {{"END",     59, 0x0000}, {"IRET",     1, 0x00CF},     /* [29] */
        {"JO",      11, 0x0070}, {"SBB",     25, 0x0018},
        {"SETAE",   23, 0x0F93}, {"SIDT",    21, 0x0108},
        {"",         0, 0x0000}, {"",         0, 0x0000},
        {"",         0, 0x0000}, {"",         0, 0x0000},
        {"",         0, 0x0000}, {"",         0, 0x0000},
        {"",         0, 0x0000}, {"",         0, 0x0000},
        {"",         0, 0x0000}, {"",         0, 0x0000}},
       {{"CDQ",      6, 0x6699}, {"CLI",      1, 0x00FA},    ' /* [OPHSH-1] */
        {"DAS",      1, 0x002F}, {"JP",      11, 0x007A},
        {"POPF",     1, 0x009D}, {"SETBE",   23, 0x0F96},
        {"WAIT",     1, 0x009B}, {"INVLPG",  21, 0x0138},
        {"",         0, 0x0000}, {"",         0, 0x0000},
        {"",         0, 0x0000}, {"",         0, 0x0000},
        {"",         0, 0x0000}, {"",         0, 0x0000},
        {"",         0, 0x0000}, {"",         0, 0x0000})
};

static void _near modgena(struct opdv *xx);
static void _near modgenb(struct opdv *xx);

/* 文字列が命令かどうかの調査 */
int isopr(char *pt) {

    char            buf[OPMAX+1],
                    *spt;
    int             ii,
                    chr,
                    sum;
    struct optbl    *wpt;

    if(strlen(pt)>OPMAX) return FALSE;
    strcpy(buf, pt);
    strupr(buf);
    spt=buf;
    sum=0;
    while(1) {
        chr=(int)*spt++;
        if(chr=='¥0') break;
        sum+=chr;
    }
    sum%=OPHSH;
    wpt=oplst[sum];
    for(ii=0; ii<OPENT; ii++) {
        if(strcmp(wpt->name, buf)==0) {
            opcod=wpt->code;
            opgrp=wpt->group;
            return TRUE;
        }
        wpt++;
    }
    return FALSE;
}

/* 1行分の構文解析 */
void srclin(void) {

    char    *pt;
    int     chr,
            sts,
            len;

    word=0L;
    lblbuf[0]=(char)'¥0';
    pt=srcbuf;
    pt=token(pt);
    chr=(int)tknbuf[0];
    if(chr==';' || chr=='¥0') {
        /* COMMENTのみの行 */
        locbuf[0]=(char)'¥0';
        mchbuf[0]=(char)'¥0';
        lstout();
        return;
    }
    if(isopr(tknbuf)==FALSE) {
        /* LABELあり */
        strcpy(lblbuf, tknbuf);
        chr=(int)*pt;
        if(chr==':') {
            pt=token(pt);
            strcat(lblbuf, tknbuf);
        }
        pt=token(pt);
    }
    chr=(int)tknbuf[0];
    if(chr!=';' && chr!='¥0') ifop(pt);
    else {
        /* LABELのみの行 */
        len=strlen(lblbuf);
        if(ifcnt==0) sts=TRUE;
        else if(ifflg[ifcnt-1]==TRUE) sts=TRUE;
        else sts=FALSE;
        if(len!=0 && sts==TRUE) {
            chr=(int)lblbuf[len-1];
            if(chr==':') {
                lblbuf[len-1]=(char)'¥0';
                lblreg(lac, LBL, SIZNER);
            }
            else errgen(36, lblbuf);
        }
        sprintf(locbuf, "%04lX    ", lac);
        mchbuf[0]=(char)'¥0';
        lstout();
    }
    if(word && passno==2) {
        /* LACの最大値と最小値の設定 */
        if(lac < lacmin) lacmin=lac;
        if(lac+word > lacmax) lacmax=lac+word;
    }
    lac+=word;
    lac&=0x0000FFFFUL;
}

/* IF条件の真理値による制御 */
void ifop(char *pt) {

    int     flg,
            chr,
            len,
            grp,
            sts;

    prefix=0;
    if(ifcnt==0) sts=TRUE;
    else if(ifflg[ifcnt-1]==TRUE) sts=TRUE;
    else sts=FALSE;
    flg=isopr(tknbuf);
    if(flg==TRUE && sts==FALSE) {
        /* IF条件が偽のときの命令 */
        if(opgrp!=58 && opgrp!=60 && opgrp!=59) {
            /* ELSE・ENDIF・END命令以外のとき */
            locbuf[0]=(char)'¥0';
            mchbuf[0]=(char)'¥0';
            lstout();
            return;
        }
    }
    else if(flg==FALSE && sts==FALSE) {
        /* IF条件が偽のときの不正な命令 */
        locbuf[0]=(char)'¥0';
        mchbuf[0]=(char)'¥0';
        lstout();
        return;
    }
    else if(flg==FALSE && sts==TRUE) {
        /* IF条件が真のときの不正な命令 */
        errgen(37, NULL);
        locbuf[0]=(char)'¥0';
        mchbuf[0]=(char)'¥0';
        lstout();
        return;
    }
    grp=opgrp;
    pt=passx(pt, FALSE);
    if(grp<50) {
        /* PROCESSOR命令のときのLABEL登録 */
        len=strlen(lblbuf);
        if(len!=0) {
            chr=(int)lblbuf[len-1];
            if(chr==':') {
                lblbuf[len-1]=(char)'¥0';
                lblreg(lac, LBL, SIZNER);
            }
            else errgen(36, lblbuf);
        }
    }
    /* OPERANDに不要な文字がないか調査 */
    pt=token(pt);
    chr=(int)tknbuf[0];
    if(chr!=';' && chr!='¥0') errgen(35, NULL);
}

/* PREFIX命令のための命令処理入り口 */
char *fxop(char *pt) {

    char    *wk;

    wk=token(pt);
    if(isopr(tknbuf)==FALSE) {
        errgen(37, NULL);
        return pt;
```

アセンブラ用語集
ディスティネーション　ソースに対して処理の対象となるオペランド

```
    }
    pt=wk;
    pt=passx(pt, TRUE);
    return pt;
}

/* オペランドの種別の調査(命令コード生成用) */
int chktyp(struct opdv *xx) {

    int   rat,
        xat;

    xat = xx->att & SPC;
    if(xat==REG) {
        rat = xx->bsr & 0xFF00;
        if(rat==BREG) return TYPBRG;
        if(rat==WREG) return TYPWRG;
        if(rat==EREG) return TYPERG;
        if(rat==SREG) {
            rat = xx->bsr & 0x00FF;
            if(rat<4) return TYPSRG;
            else return TYPSRX;
        }
        return TYPXRG;
    }
    if(xx->bsr) {
        rat = xx->bsr & 0xFF00;
        if(rat==WREG) return TYPIND;
        return TYPINX;
    }
    if(xx->ixr) {
        rat = xx->ixr & 0xFF00;
        if(rat==WREG) return TYPIND;
        return TYPINX;
    }
    if(xx->siz == SIZFAR) return TYPFAR;
    if(xat == LBL) return TYPLBL;
    if(xat == VAR) return TYPVAR;
    return TYPNUM;
}

/* OPERANDの値からmdregmem BYTEとscixrbsr BYTEを生成 */
void modgen(struct opdv *xx) {

    if(prctyp<=2) modgena(xx);
    else modgenb(xx);
}

/* 286以下のCPUの場合 */
static void _near modgena(struct opdv *xx) {

    int             md,
                    reg,
                    mem,
                    bas,
                    inx,
                    xat,
                    typ;
    unsigned long val;

    opasz=0;
    opovr=0;
    opmod=0;
    opsib=0;
    opdisp=0UL;
    xat = xx->att;
    if(passno!=3) val = xx->p1val;
    else val = xx->p2val;
    typ=chktyp(xx);
    reg = xx->ovr;
    if(reg!=0 && (reg & 0x00FF)<4) opovr=segcnv(reg);
    if(typ==TYPBRG || typ==TYPWRG || typ==TYPSRG)
        opmod=(xx->bsr & 0x00FF)|0x00C0;
    else if(typ==TYPIND) {
        md=0x0000;                          /* 0[xx] */
        if((xat&DEF)==0) md=0x0080;         /* d16[xx] */
        else {
            if(val!=0L) {
                if(val>=BYTMIN || val<=BYTMAX) md=0x0040; /* d8[xx] */
                else md=0x0080;             /* d16[xx] */
            }
        }
        if(xx->ixr == 0) {
            bas = xx->bsr & 0x00FF;
            if(bas==3) mem=0x0007;          /* xx[BX] */
            else if(bas==5) {
                if(md==0x0000) md==0x0040;  /* 0[BP] */
                mem=0x0006;                 /* xx[BP] */
            }
            else if(bas==6) mem=0x0004;     /* xx[SI] */
            else mem=0x0005;                /* xx[DI] */
        }
        else {
            bas = xx->bsr & 0x00FF;
            inx = xx->ixr & 0x00FF;
            if(bas==3) {
                if(inx==6) mem=0x0000;      /* [BX+SI] */
                else mem=0x0001;            /* [BX+DI] */
            }
            else {
                if(inx==6) mem=0x0002;      /* [BP+SI] */
                else mem=0x0003;            /* [BF+DI] */
            }
        }
        opmod=md|mem;
        opdisp=val;
    }
    else if(typ==TYPNUM || typ==TYPLBL || typ==TYPVAR) {
        opmod=0x0006;                       /* DA16 */
        opdisp=val;
    }
}

/* 386以上のCPUの場合 */
static void _near modgenb(struct opdv *xx) {

    int             md,
                    reg,
                    mem,
                    bas,
                    inx,
                    sc,
                    xat,
                    typ;
    unsigned long val;

    opasz=0;
    opovr=0;
    opmod=0;
    opsib=0;
    opdisp=0UL;
    xat = xx->att;
    if(passno!=3) val = xx->p1val;
    else val = xx->p2val;
    typ=chktyp(xx);
    reg = xx->ovr;
    if(reg!=0) opovr=segcnv(reg);
    if(typ==TYPBRG || typ==TYPWRG || typ==TYPSRG || typ==TYPSRX ||
        typ==TYPXRG) opmod=(xx->bsr & 0x00FF)|0x00C0;
    else if(typ==TYPERG) opmod=(xx->bsr & 0x00FF)|0x00C0;
    else if(typ==TYPIND) {
        md=0x0000;                          /* 0[xx] */
        if((xat&DEF)==0) md=0x0080;         /* d16[xx] */
        else {
            if(val!=0L) {
                if(val>=BYTMIN || val<=BYTMAX) md=0x0040; /* d8[xx] */
                else md=0x0080;             /* d16[xx] */
            }
        }
        if(xx->ixr == 0) {
            bas = xx->bsr & 0x00FF;
            if(bas==3) mem=0x0007;          /* xx[BX] */
            else if(bas==5) {
                if(md==0x0000) md==0x0040;  /* 0[BP] */
                mem=0x0006;                 /* xx[BP] */
            }
            else if(bas==6) mem=0x0004;     /* xx[SI] */
            else mem=0x0005;                /* xx[DI] */
        }
        else {
            bas = xx->bsr & 0x00FF;
            inx = xx->ixr & 0x00FF;
            if(bas==3) {
                if(inx==6) mem=0x0000;      /* [BX+SI] */
                else mem=0x0001;            /* [BX+DI] */
            }
            else {
                if(inx==6) mem=0x0002;      /* [BP+SI] */
                else mem=0x0003;            /* [BP+DI] */
            }
        }
        opmod=md|mem;
        opdisp=val;
    }
    else if(typ==TYPINX) {
        md=0x0000;
        sc=0x0000;
        bas=0x0000;
        inx=0x0000;
        opasz=0x0067;
        md=0x0000;
        if(xx->ixr) {                           /* xx[bsr+ixr*n] */
            if(xx->scl == 2) sc=0x0040;          /* ixr*2 */
            else if(xx->scl == 4) sc=0x0080;     /* ixr*4 */
            else if(xx->scl == 8) sc=0x00C0;     /* ixr*8 */
            inx=(xx->ixr & 0x00FF)<<3;
            bas = xx->bsr & 0x00FF;
            mem=0x0004;
        }
        else if((xx->bsr & 0x00FF)==4) {        /* xx[SP] */
            mem=0x0004;
            bas=0x0004;
            inx=0x0020;
        }
        else mem = xx->bsr & 0x00FF;
        if((xat&DEF)==0) md=0x0030;
            }
        }
        else if(typ==TYPLBL) {
            lx=expval.p1val-lac-2L;
            if(lx>=BYTMIN || lx<=BYTMAX) {
                ly=expval.p2val-lac-2L;
                mchbvst(0x00EB);
                mchbvst((int)lx);
            }
            else {
                mchbvst(0x00E9);
                lx=expval.p2val-lac-3L;
                mchwvst((int)lx);
            }
        }
        else if(typ==TYPWRG) mr16(0x00FF, 0x0020, &expval);
        else if(typ==TYPVAR || typ==TYPIND) {
            if(expval.siz == SIZDWD) mr16(0x00FF, 0x0028, &expval);
            else {
                mr16(0x00FF, 0x0020, &expval);
                if(expval.siz!=SIZWRD) errgen(45, NULL);
            }
        }
        else if(typ==TYPFAR)
```


アセンブラ用語集
データセグメント　プログラムで使われるデータが格納されるアドレスのセグメント.

```
            n32(0x00EA, (int)expval.p2val, (int)expval.aseg);
        else errgen(46, NULL);
    }
    else {
        if(sht) {
            mchbvst(0x00EB);
            lx=expval.p2val-lac-2L;
            if(lx>=BYTMIN || lx<=BYTMAX) mchbvst((int)lx);
            else {
                errgen(47, NULL);
                mchbvst(0);
            }
        }
        else if(typ==TYPLBL) {
            lx=expval.p1val-lac-2L;
            if(lx>=BYTMIN || lx<=BYTMAX) {
                ly=expval.p2val-lac-2L;
                mchbvst(0x00EB);
                mchbvst((int)lx);
            }
            else {
                mchbvst(0x00E9);
                lx=expval.p2val-lac-3L;
                mchwvst((int)lx);
            }
        }
        else if(typ==TYPWRG) mr32(0x00FF, 0x0020, &expval);
        else if(typ==TYPERG) {
            mchbvst(0x0066);
            mr32(0x00FF, 0x0020, &expval);
        }
        else if(typ==TYPVAR || typ==TYPIND || typ==TYPINX) {
            if(expval.siz==SIZDWD) mr32(0x00FF, 0x0028, &expval);
            else if(expval.siz==SIZFWD) {
                mchbvst(0x0066);
                mr32(0x00FF, 0x0028, &expval);
            }
            else {
                mr32(0x00FF, 0x0020, &expval);
                if(expval.siz!=SIZWRD) errgen(45, NULL);
            }
        }
        else if(typ==TYPFAR)
            n32(0x00EA, (int)expval.p2val, (int)expval.aseg);
        else errgen(46, NULL);
    }

    if((xx->att & SPC)!=REG) return FALSE;
    if(xx->bsr != EREG+0) return FALSE;
    return TRUE;
}

/* SHORTがあるかないかの調査 */
int issht(char *pt) {

    token(pt);
    if(stricmp(tknbuf, "SHORT")) return FALSE;
    return TRUE;
}

/* OPERANDが終わりかどうかの調査 */
int isend(char *pt) {

    int     chr;

    token(pt);
    chr=(int)tknbuf[0];
    if(chr==';' || chr=='¥0') return TRUE;
    return FALSE;
}

/* 次のOPERANDへ処理を進める */
char *nxtod(char *pt) {

    char    *wk;
    int     chr;

    while(1) {
        wk=token(pt);
        chr=(int)tknbuf[0];
        if(chr==';' || chr=='¥0') {
            errgen(40, NULL);
            break;
        }
        pt=wk;
        if(chr==',') break;
    }
    return pt;
}

/* L16形式のOBJECTを出力する */

void m16(int mch, struct opdv *xx) {

    if(xx->ovr) xx->ovr=segcnv(xx->ovr);
    mchbvst(xx->ovr);
    mchbvst(mch);
    mchwvst((int)xx->p2val);
}

/* D16形式のOBJECTを出力する */
void n16(int mch, int ofs) {

    mchbvst(mch);
    mchwvst(ofs);
}

/* L32形式のOBJECTを出力する */
void n32(int mch, int ofs, int sg) {

    mchbvst(mch);
    mchwvst(ofs);
    mchwvst(sg);
}

/* 16BIT MODE mrr形式のOBJECTを出力 */
void mr16(int mch, int reg, struct opdv *xx) {

    int     md,
            mem;

    modgen(xx);
    if(opovr) mchbvst(opovr);
    if(mch & 0xFF00) mchbvst(mch>>8);
    mchbvst(mch);
    mchbvst(opmod|reg);
    md=opmod&0x00C0;
    mem=opmod&0x0007;
    if(md==0x0080) mchwvst((int)opdisp);
    else if(md==0x0040) mchbvst((int)opdisp);
    else if(md==0x0000 && mem==0x0006) mchwvst((int)opdisp);
}

/* 32BIT MODE mrr形式のOBJECTを出力 */
void mr32(int mch, int reg, struct opdv *xx) {

    int     md,
            mem,
            inx,
            bas;

    modgen(xx);
    if(opasz) mchbvst(opasz);
    if(opovr) mchbvst(opovr);
    if(mch&0xFF00) mchbvst(mch>>8);
    mchbvst(mch);
    mchbvst(opmod|reg);
    if(opasz) {
        md=opmod&0x00C0;
        mem=opmod&0x0007;
        inx=opsib&0x0038;
        bas=opsib&0x0007;
        if(md!=0x00C0 && mem==0x0004) mchbvst(opsib);
        if(md==0x0040) mchbvst((int)opdisp);
        else if(md==0x0080) mchlvst(opdisp);
        else if(md==0x0000 && mem==0x0005) mchlvst(opdisp);
    }
    else {
        md=opmod&0x00C0;
        mem=opmod&0x0007;
        if(md==0x0080) mchwvst((int)opdisp);
        else if(md==0x0040) mchbvst((int)opdisp);
        else if(md==0x0000 && mem==0x0006) mchwvst((int)opdisp);
    }
}

/* PASS1未定義時のDEFAULT属性をLBLとする */
int typudl(struct opdv *xx) {

    int     typ;

    typ=chktyp(xx);
    if(passno==1 && typ==TYPNUM && (xx->att & DEF)==0) return TYPLBL;
    return typ;
}

/* PASS1未定義時のDEFAULT属性をVARとする(WORD/DWORD) */
int typudv(struct opdv *xx) {

    int     typ;

    typ=chktyp(xx);
    if(passno==1 && typ==TYPNUM && (xx->att & DEF)==0) {
        if(prctyp<=2) xx->siz=SIZWRD;
        else xx->siz=SIZDWD;
        return TYPVAR;
    }
    return typ;
}

/* PASS1未定義時のDEFAULT属性をVARとする(DWORD/FWORD) */
int typudf(struct opdv *xx) {

    int     typ;

    typ=chktyp(xx);
    if(passno==1 && typ==TYPNUM && (xx->att & DEF)==0) {
        if(prctyp<=2) xx->siz=SIZDWD;
        else xx->siz=SIZFWD;
        return TYPVAR;
    }
    return typ;
}

/* 値が符号付き8BITの範囲かどうか調査する */
int chkis8(struct opdv *xx) {

    if((xx->att & DEF)==0) return FALSE;    /* 未定義のときは範囲外 */
    if(xx->p1val>=BYTMIN || xx->p1val<=BYTMAX) return TRUE;
    return FALSE;
}

/* 値が符号付き16BITの範囲かどうか調査する */
int chkis16(struct opdv *xx) {

    if((xx->att & DEF)==0) return FALSE;    /* 未定義のときは範囲外 */
    if(xx->p1val>=WRDMIN || xx->p1val<=WRDMAX) return TRUE;
    return FALSE;
}
```

アセンブラ用語集
デバッグ　プログラムの誤りを訂正する作業．デバッガを使わず，目でプログラムの流れを追いながらする作業をハンドデバッグという．

1.オペランドの個数
2.使用可能な最下位のプロセッサ
3.使用できるオペランド

そして，最終的には同じグループに属する命令は，同じ方法で機械コードが生成できることを原則としている．この結果，プロセッサ命令は49グループ，ディレクティブは1命令1グループで14グループに分かれている．プロセッサ命令も1命令1グループのものが多いが，あまり凝ったグループ分けをするより単純な分け方をしたほうが後のプログラミングが楽なので，このようなグループ分けに落ち着いている．

6-2 構文解析

構文解析は，pass1，pass2およびpass3の各関数から直接呼び出されるsrclin関数が行う．ここで行われる構文解析は，シンボルの有無と命令の有無である．すなわち，srclin関数では，何も書かれていない行とコメントだけの行，ラベルだけの行およびシンボルと命令あるいは命令の書かれている行の3つに分類して処理を決める．

命令が書かれているかどうかのチェックは，isopr関数を呼び出して調べる．isopr関数は，命令表のニモニックとソースプログラムの文字列を大文字小文字の区別なく比較し，もし一致すれば命令表から命令のグループをopgrpに，命令の識別コードをopcodに格納してTRUEを返す．そうでなければ，何もしないでFALSEを返す．srclin関数は，TRUEが返されれば命令があるものと判断し，FALSEが返されればシンボルがあるものと判断する．

何も書かれていない行とコメントだけの行は，リストファイルに出力を行って処理終了となる．ラベルだけの書かれている行（ラベルの後にコメントが書かれている場合を含む）では，IFブロックの条件の真偽を判定した上で，真であればlacの値を値とし，属性をラベルとしてシンボルテーブルに登録する．IFブロックの条件が偽であればラベルとしての登録は行わない．この後リストファイルに行の内容を出力して処理を終了する．シンボルと命令あるいは命令が書かれている場合には，ifop関数を呼び出して，IFブロックの条件の真偽のチェックへ処理を進める．

ifop関数から復帰すると，機械コードの生成が終了し生成した機械コードのバイト長がword変数に残っているので，lacのカウントアップとlacminおよびlacmaxの更新処理を行う．

6-3 IFブロックの条件チェック

ifブロックの条件チェックはifop関数で行う．処理している命令がIFブロックの外にあるならifcnt変数は0であり，無条件にpassx関数を呼び出して機械コードの生成やディレクティブの処理を行う．そうでないなら，ifflg配列を調べて現在の条件が真か偽かを調べる．現在の条件が真のときは，passx関数を呼び出す．現在の条件が偽のときは，opgrpから命令がELSE,ENDIFあるいはENDでないか調べ，これらの命令のときのみpassx関数を呼び出す．この3つの命令以外のときは，行をリストファイルに出力するだけで処理を終了する．

passx関数から戻ると，opgrpにより命令がプロセッサ命令であったかディレクティブであったかを判断し，プロセッサ命令でラベルがある場合にはラベルの登録を行う．ディレクティブの場合には，個別にシンボルの処理が必要なので，ディレクティブの処理関数の中でシンボルの処理を行っている．

6-4 機械コード生成の手順

プログラム6（PX86.C）の最初の関数passxは，isopr関数のセットしたopgrpにより命令のグループ別に機械コードを生成する関数op01～op63までを呼び出す．この呼び出しに先だってmchopn関数を呼び出し，アセンブリリストの1行の初期化処理を行う．また，機械コードの生成が終了すると，mchcls関数を呼び出して1行の後始末を行う．ただし，第2引数のcntがFALSEでない場合は，プリフィックス命令の処理から呼び出された場合であるので，プリフィックス命令とそれに引き続く命令を1行として処理するためにmchopnとmchcls関数は呼び出さない．

機械コード生成の関数の処理グループごとに少しずつ異なっているが，共通点を疑似的なC言語風にまとめれば

アセンブラ用語集
ニーモニック　2進数の集まりであるマシンコードを人間が理解し易いように簡略化された単語をニーモニックという．

図6の通りである。

図6の中で、オペランドの誤りと使用できない命令の場合は、op01～op49の関数は機械コードを生成しない。errgen関数を呼び出して、エラーメッセージを出力するのみである。表1～表4と組み合わせてプログラムを読んでみるとわかりやすいと思う。

op50～op63の関数は、ディレクティブの処理で、こちらは関数毎に処理が異なっている。これらの関数はあまり複雑なものはないので、読者自身でプログラムを読んで確かめてほしい。

図6

```
機械コードの生成処理() {
        必要な数のオペランドの値を求める;
        各オペランドのタイプを調べる;
        if(現在のCPUで使用できる命令) {
                if(オペランドのタイプが正しい組み合わせ1)
                        組み合わせ1の機械コード出力;
                else if(オペランドのタイプが正しい組み合わせ2)
                        組み合わせ2の機械コード出力;
                ...............................................
                ...............................................
                else オペランドの誤り;
        }
        else 使用できない命令;
}
```

リスト6 PX86.C

```
/******************************************/
/*  各命令の処理(AS86)                     */
/******************************************/

#include <STDLIB.H>
#include <STDIO.H>
#include <STRING.H>
#include "AS86.H"

static struct opdv
      opd1,   /* 第1OPERANDの値 */
      opd2,   /* 第2OPERANDの値 */
      opd3;   /* 第3OPERANDの値 */

/* OPERANDのない命令 */
static char *op01(char *pt);
static char *op02(char *pt);
static char *op03(char *pt);
static char *op04(char *pt);
static char *op05(char *pt);
static char *op06(char *pt);
/* 1-OPERANDの命令 */
static char *op07(char *pt);
static char *op08(char *pt);
static char *op09(char *pt);
static char *op10(char *pt);
static char *op11(char *pt);
static char *op12(char *pt);
static char *op13(char *pt);
static char *op14(char *pt);
static char *op15(char *pt);
static char *op16(char *pt);
static char *op17(char *pt);
static char *op18(char *pt);
static char *op19(char *pt);
static char *op20(char *pt);
static char *op21(char *pt);
static char *op22(char *pt);
static char *op23(char *pt);
static char *op24(char *pt);
/* 2-OPERANDの命令 */
static char *op25(char *pt);
static char *op25a(char *pt);
static char *op25b(char *pt);
static char *op26(char *pt);
static char *op26a(char *pt);
static char *op26b(char *pt);
static char *op27(char *pt);
static char *op28(char *pt);
static char *op29(char *pt);
static char *op30(char *pt);
static char *op31(char *pt);
static char *op32(char *pt);
static char *op32a(char *pt);
static char *op32b(char *pt);
static char *op33(char *pt);
static char *op34(char *pt);
static void op34a(void);
static void op34b(void);
static void op34c(void);
static char *op35(char *pt);
static char *op36(char *pt);
static char *op37(char *pt);
static char *op38(char *pt);
static char *op39(char *pt);
static char *op40(char *pt);
static char *op41(char *pt);
static char *op42(char *pt);
static char *op43(char *pt);
static char *op44(char *pt);
static char *op45(char *pt);
static char *op46(char *pt);
static char *op47(char *pt);
/* 3-OPERANDの命令 */
static char *op48(char *pt);
static void op48a(void);
static void op48b(void);
static void op48c(void);
static char *op49(char *pt);
/* ディレクティブ */
static char *op50(char *pt);
static char *op51(char *pt);
static char *op52(char *pt);
static char *op53(char *pt);
static char *op54(char *pt);
static char *op55(char *pt);
static char *op56(char *pt);
static char *op57(char *pt);
static char *op58(char *pt);
static char *op59(char *pt);
static char *op60(char *pt);
static char *op61(char *pt);
static char *op62(char *pt);
static char *op63(char *pt);

/* 命令GROUPによる処理の切り分け */
char *passx(char *pt, int cnt) {

      if(cnt==FALSE) mchopn();
      switch(opgrp) {
      case  1:
            pt=op01(pt);
            break;
      case  2:
            pt=op02(pt);
            break;
      case  3:
            pt=op03(pt);
            break;
      case  4:
            pt=op04(pt);
            break;
      case  5:
            pt=op05(pt);
            break;
      case  6:
            pt=op06(pt);
            break;
      case  7:
            pt=op07(pt);
            break;
      case  8:
            pt=op08(pt);
            break;
      case  9:
            pt=op09(pt);
            break;
      case 10:
            pt=op10(pt);
            break;
      case 11:
            pt=op11(pt);
            break;
      case 12:
            pt=op12(pt);
            break;
      case 13:
            pt=op13(pt);
            break;
      case 14:
            pt=op14(pt);
            break;
      case 15:
            pt=op15(pt);
            break;
      case 16:
            pt=op16(pt);
            break;
      case 17:
            pt=op17(pt);
```

アセンブラ用語集

ハードウェア割り込み　CPUはINTR端子に接続されたハードウェアからの割り込み信号を受けて、割り込み処理ルーチンに制御を移す。RS232Cなどの制御に使われる。

```
            break;
    case 18:
        pt=op18(pt);
        break;
    case 19:
        pt=op19(pt);
        break;
    case 20:
        pt=op20(pt);
        break;
    case 21:
        pt=op21(pt);
        break;
    case 22:
        pt=op22(pt);
        break;
    case 23:
        pt=op23(pt);
        break;
    case 24:
        pt=op24(pt);
        break;
    case 25:
        pt=op25(pt);
        break;
    case 26:
        pt=op26(pt);
        break;
    case 27:
        pt=op27(pt);
        break;
    case 28:
        pt=op28(pt);
        break;
    case 29:
        pt=op29(pt);
        break;
    case 30:
        pt=op30(pt);
        break;
    case 31:
        pt=op31(pt);
        break;
    case 32:
        pt=op32(pt);
        break;
    case 33:
        pt=op33(pt);
        break;
    case 34:
        pt=op34(pt);
        break;
    case 35:
        pt=op35(pt);
        break;
    case 36:
        pt=op36(pt);
        break;
    case 37:
        pt=op37(pt);
        break;
    case 38:
        pt=op38(pt);
        break;
    case 39:
        pt=op39(pt);
        break;
    case 40:
        pt=op40(pt);
        break;
    case 41:
        pt=op41(pt);
        break;
    case 42:
        pt=op42(pt);
        break;
    case 43:
        pt=op43(pt);
        break;
    case 44:
        pt=op44(pt);
        break;
    case 45:
        pt=op45(pt);
        break;
    case 46:
        pt=op46(pt);
        break;
    case 47:
        pt=op47(pt);
        break;
    case 48:
        pt=op48(pt);
        break;
    case 49:
        pt=op49(pt);
        break;
    case 50:
        pt=op50(pt);
        break;
    case 51:
        pt=op51(pt);
        break;
    case 52:
        pt=op52(pt);
        break;
    case 53:
        pt=op53(pt);
        break;
    case 54:
        pt=op54(pt);
        break;
    case 55:
        pt=op55(pt);
        break;
    case 56:
        pt=op56(pt);
        break;
    case 57:
        pt=op57(pt);
        break;
    case 58:
        pt=op58(pt);
        break;
    case 59:
        pt=op59(pt);
        break;
    case 60:
        pt=op60(pt);
        break;
    case 61:
        pt=op61(pt);
        break;
    case 62:
        pt=op62(pt);
        break;
    case 63:
        pt=op63(pt);
        break;
    }
    if(cnt==FALSE) mchcls();
    return pt;
}

/* AAA, AAD, AAM, AAS, CBW, CLC, CLD, CLI, CMC, CMPSB, CMPSW,
   CWD, DAA, DAS, HLT, INTO, IRET, LAHF, LODSB, LODSW, MOVSB,
   MOVSW, NOP, POPF, PUSHF, SAHF, SCASB, SCASW, STC, STD, STI,
   STOSB, STOSW, WAIT */
static char *op01(char *pt) {

    if(opcod & 0xFF00) mchbvst((opcod>>8)&0x00FF);
    mchbvst(opcod & 0x00FF);
    return pt;
}

/* LOCK, REP, REPE, REPNE, REPNZ, REPZ */
static char *op02(char *pt) {

    mchbvst(opcod);
    prefix=opcod;
    pt=fxop(pt);
    return pt;
}

/* INSB, INSW, LEAVE, OUTSB, OUTSW, POPA, PUSHA */
static char *op03(char *pt) {

    if(prctyp>=1) mchbvst(opcod);
    else errgen(41, NULL);
    return pt;
}

/* CLTS, INVD, WBINVD */
static char *op04(char *pt) {

    if(opcod==0x0F06) {     /* CLTS */
        if(prctyp>=2) {
            mchbvst(0x000F);
            mchbvst(0x0006);
        }
        else errgen(42, NULL);
    }
    else {                  /* INVD, WBINVD */
        if(prctyp>=4) {
            mchbvst(opcod>>8);
            mchbvst(opcod);
        }
        else errgen(44, NULL);
    }
    return pt;
}

/* ASIZE, OSIZE */
static char *op05(char *pt) {

    if(prctyp>=3) mchbvst(opcod);
    else errgen(43, NULL);
    prefix=opcod;
    pt=fxop(pt);
    return pt;
}

/* CDQ, CMPSD, CWDE, INSD, IRETD, LODSD, MOVSD, OUTSD, POPAD, POPFD,
   PUSHAD, PUSHFD, SCASD, STOSD */
static char *op06(char *pt) {

    if(prctyp>=3) {
        mchbvst(opcod>>8);  /* OSIZE */
        mchbvst(opcod);     /* OPCODE */
    }
    else errgen(43, NULL);
    return pt;
}

/* CALL */
static char *op07(char *pt) {

    int             typ;
    unsigned long vl;

    pt=expr(pt);
    typ=typudf(&expval);
    if(prctyp<=2) {
        if(typ==TYPLBL) {
            vl=expval.p2val-lac-3L;
            n16(0x00E8, (int)vl);
        }
        else if(typ==TYPWRG) mr16(0x00FF, 0x0010, &expval);
        else if(typ==TYPVAR || typ==TYPIND) {
            if(expval.siz==SIZDWD) mr16(0x00FF, 0x0018, &expval);
            else {
                mr16(0x00FF, 0x0010, &expval);
                if(expval.siz!=SIZWRD) errgen(45, NULL);
            }
        }
        else if(typ==TYPFAR) n32(0x009A, (int)vl, (int)expval.aseg);
        else errgen(46, NULL);
    }
    else {
        if(typ==TYPLBL) {
            vl=expval.p2val-lac-3L;
            n16(0x00E8, (int)vl);
        }
        else if(typ==TYPWRG) mr32(0x00FF, 0x0010, &expval);
        else if(typ==TYPERG) {
            mchbvst(0x0066);
```

アセンブラ用語集
パリティフラグ　演算の結果，下位8ビットの中でセットされているビットが偶数ならセットされ，奇数ならリセットされるフラグ．エラーチェックに使われる．

```
                mr32(0x00FF, 0x0010, &expval);
            }
            else if(typ==TYPVAR || typ==TYPIND || typ==TYPINX) {
                if(expval.siz == SIZDWD) mr32(0x00FF, 0x0018, &expval);
                else if(expval.siz == SIZFWD) {
                    mchbvst(0x0066);
                    mr32(0x00FF, 0x0018, &expval);
                }
                else {
                    mr32(0x00FF, 0x0010, &expval);
                    if(expval.siz!=SIZWRD) errgen(45, NULL);
                }
            }
            else if(typ == TYPFAR) {
                vl=expval.p2val;
                n32(0x009A, (int)vl, (int)expval.aseg);
            }
        }
        return pt;
}

/* INC, DEC */
static char *op08(char *pt) {

        int     typ,
                reg;

        pt=expr(pt);
        typ=typudv(&expval);
        reg=opcod&0x0008;
        if(prctyp<=2) {
            if(typ==TYPWRG) mchbvst(opcod+(expval.bsr & 0x00FF));
            else if(typ==TYPBRG) mr16(0x00FE, reg, &expval);
            else if(typ==TYPVAR || typ==TYPIND) {
                if(expval.siz == SIZBYT) mr16(0x00FE, reg, &expval);
                else {
                    mr16(0x00FF, reg, &expval);
                    if(expval.siz!=SIZWRD) errgen(45, NULL);
                }
            }
            else errgen(46, NULL);
        }
        else {
            if(typ==TYPWRG) mchbvst(opcod+(expval.bsr & 0x00FF));
            else if(typ==TYPERG) {
                mchbvst(0x0066);
                mchbvst(opcod + expval.bsr & 0x00FF);
            }
            else if(typ==TYPBRG) mr32(0x00FE, reg, &expval);
            else if(typ==TYPVAR || typ==TYPIND || typ==TYPINX) {
                if(expval.siz==SIZBYT) mr32(0x00FE, reg, &expval);
                else if(expval.siz==SIZDWD) {
                    mchbvst(0x0066);
                    mr32(0x00FF, reg, &expval);
                }
                else {
                    mr32(0x00FF, reg, &expval);
                    if(expval.siz!=SIZWRD) errgen(45, NULL);
                }
            }
            else errgen(46, NULL);
        }
        return pt;
}

/* DIV, IDIV, MUL, NEG, NOT */
static char *op09(char *pt) {

        int     typ;

        pt=expr(pt);
        typ=typudv(&expval);
        if(prctyp<=2) {
            if(typ==TYPBRG) mr16(0x00F6, opcod, &expval);
            else if(typ==TYPWRG) mr16(0x00F7, opcod, &expval);
            else if(typ==TYPVAR || typ==TYPIND) {
                if(expval.siz == SIZBYT) mr16(0x00F6, opcod, &expval);
                else {
                    mr16(0x00F7, opcod, &expval);
                    if(expval.siz!=SIZWRD) errgen(45, NULL);
                }
            }
            else errgen(46, NULL);
        }
        else {
            if(typ==TYPBRG) mr32(0x00F6, opcod, &expval);
            else if(typ==TYPWRG) mr32(0x00F7, opcod, &expval);
            else if(typ==TYPERG) {
                mchbvst(0x0066);
                mr32(0x00F7, opcod, &expval);
            }
            else if(typ==TYPVAR || typ==TYPIND || typ==TYPINX) {
                if(expval.siz == SIZBYT) mr32(0x00F6, opcod, &expval);
                else if(expval.siz == SIZDWD) {
                    mchbvst(0x0066);
                    mr32(0x00F7, opcod, &expval);
                }
                else {
                    mr32(0x00F7, opcod, &expval);
                    if(expval.siz!=SIZWRD) errgen(45, NULL);
                }
            }
            else errgen(46, NULL);
        }
        return pt;
}

/* INT */
static char *op10(char *pt) {

        int             typ;
        unsigned long vl;

        pt=expr(pt);
        typ=chktyp(&expval);
        vl=expval.p2val;
        if(typ==TYPNUM && vl==3UL) mchbvst(0x00CC);
        else if(typ==TYPNUM) {
            mchbvst(0x00CD);
            mchbvst((int)vl);
            if(vl>=0x000000FFL) errgen(47, NULL);
        }
        else errgen(46, NULL);
        return pt;
}

/* Jcc */
static char *op11(char *pt) {

        int             sht,
                        typ;
        unsigned long lx,
                        ly;

        sht=issht(pt);
        if(sht) pt=token(pt);
        pt=expr(pt);
        typ=typudl(&expval);
        if(typ==TYPLBL) {
            if(prctyp<=2) {
                lx=expval.p2val-lac-2;
                if(lx>=BYTMIN || lx<=BYTMAX) {
                    mchbvst(opcod);
                    mchbvst((int)lx);
                }
                else {
                    errgen(47, NULL);
                    mchbvst(opcod);
                    mchbvst(0);
                }
            }
            else {
                if(sht) {
                    lx=expval.p2val-lac-2;
                    if(lx>=BYTMIN || lx<=BYTMAX) {
                        mchbvst(opcod);
                        mchbvst((int)lx);
                    }
                    else {
                        errgen(47, NULL);
                        mchbvst(opcod);
                        mchbvst(0);
                    }
                }
                else if((expval.att & DEF)==0) {
                    mchbvst(0x000F);
                    mchbvst(opcod+0x0010);
                    mchwvst(0);
                }
                else {
                    lx=expval.p1val-lac-2L;
                    if(lx>=BYTMIN || lx<=BYTMAX) {
                        ly=expval.p2val-lac-2L;
                        mchbvst(opcod);
                        mchbvst((int)ly);
                    }
                    else {
                        ly=expval.p2val-lac-4L;
                        mchbvst(0x000F);
                        mchbvst(opcod+0x0010);
                        mchwvst((int)ly);
                    }
                }
            }
        }
        else errgen(46, NULL);
        return pt;
}

/* JECXZ */
static char *op12(char *pt) {

        int             typ;
        unsigned long lx;

        pt=expr(pt);
        typ=typudl(&expval);
        if(prctyp>=3) {
            if(typ=TYPLBL) {
                lx=expval.p2val-lac-3L;
                mchbvst(0x0067);
                mchbvst(0x00E3);
                if(lx>=BYTMIN || lx<=BYTMAX) mchbvst((int)lx);
                else {
                    errgen(47, NULL);
                    mchbvst(0);
                }
            }
            else errgen(46, NULL);
        }
        else errgen(43, NULL);
        return pt;
}

/* JMP */
static char *op13(char *pt) {

        int             typ,
                        sht;
        unsigned long lx,
                        ly;

        sht=issht(pt);
        if(sht!=FALSE) pt=token(pt);
        pt=expr(pt);
        typ=typudf(&expval);
        if(prctyp<=2) {
            if(sht) {
                mchbvst(0x00EB);
                lx=expval.p2val-lac-2L;
                if(lx>=BYTMIN || lx<=BYTMAX) mchbvst((int)lx);
                else {
                    errgen(47, NULL);
                    mchbvst(0);
                }
            }
            else if(typ==TYPLBL) {
                lx=expval.p1val-lac-2L;
                if(lx>=BYTMIN || lx<=BYTMAX) {
                    ly=expval.p2val-lac-2L;
                    mchbvst(0x00EB);
                    mchbvst((int)lx);
                }
                else {
                    mchbvst(0x00E9);
                    lx=expval.p2val-lac-3L;
```

アセンブラ用語集

バイト　8ビットを1バイトといい，コンピュータでもっともよく使われる単位.

```
                mchwvst((int)lx);
            }
        }
        else if(typ==TYPWRG) mr16(0x00FF, 0x0020, &expval);
        else if(typ==TYPVAR || typ==TYPIND) {
            if(expval.siz == SIZDWD) mr16(0x00FF, 0x0028, &expval);
            else {
                mr16(0x00FF, 0x0020, &expval);
                if(expval.siz!=SIZWRD) errgen(45, NULL);
            }
        }
        else if(typ==TYPFAR)
            n32(0x00EA, (int)expval.p2val, (int)expval.aseg);
        else errgen(46, NULL);
    }
    else {
        if(sht) {
            mchbvst(0x00EB);
            lx=expval.p2val-lac-2L;
            if(lx>=BYTMIN || lx<=BYTMAX) mchbvst((int)lx);
            else {
                errgen(47, NULL);
                mchbvst(0);
            }
        }
        else if(typ==TYPLBL) {
            lx=expval.p1val-lac-2L;
            if(lx>=BYTMIN || lx<=BYTMAX) {
                ly=expval.p2val-lac-2L;
                mchbvst(0x00EB);
                mchbvst((int)lx);
            }
            else {
                mchbvst(0x00E9);
                lx=expval.p2val-lac-3L;
                mchwvst((int)lx);
            }
        }
        else if(typ==TYPWRG) mr32(0x00FF, 0x0020, &expval);
        else if(typ==TYPERG) {
            mchbvst(0x0066);
            mr32(0x00FF, 0x0020, &expval);
        }
        else if(typ==TYPVAR || typ==TYPIND || typ==TYPINX) {
            if(expval.siz==SIZDWD) mr32(0x00FF, 0x0028, &expval);
            else if(expval.siz==SIZFWD) {
                mchbvst(0x0066);
                mr32(0x00FF, 0x0028, &expval);
            }
            else {
                mr32(0x00FF, 0x0020, &expval);
                if(expval.siz!=SIZWRD) errgen(45, NULL);
            }
        }
        else if(typ==TYPFAR)
            n32(0x00EA, (int)expval.p2val, (int)expval.aseg);
        else errgen(46, NULL);
    }
    return pt;
}

/* LODS */
static char *op14(char *pt) {

    pt=expr(pt);
    expval.ovr=segcnv(expval.ovr);
    if(prctyp<=2) {
        if(expval.ovr) mchbvst(expval.ovr);
        if(expval.siz == SIZBYT) mchbvst(0x00AC);
        else {
            mchbvst(0x00AD);
            if(expval.siz!=SIZWRD) errgen(45, NULL);
        }
    }
    else {
        if(expval.siz==SIZDWD) {
            mchbvst(0x0066);
            if(expval.ovr) mchbvst(expval.ovr);
            mchbvst(0x00AD);
        }
        else if(expval.siz==SIZBYT) {
            if(expval.ovr) mchbvst(expval.ovr);
            mchbvst(0x00AC);
        }
        else {
            if(expval.ovr) mchbvst(expval.ovr);
            mchbvst(0x00AD);
            if(expval.siz != SIZWRD) errgen(45, NULL);
        }
    }
    return pt;
}

/* SCAS, STOS */
static char *op15(char *pt) {

    pt=expr(pt);
    if(prctyp<=2) {
        if(expval.ovr) errgen(49, NULL);
        if(expval.siz==SIZBYT) mchbvst(opcod);
        else {
            mchbvst(opcod+1);
            if(expval.siz != SIZWRD) errgen(45, NULL);
        }
    }
    else {
        if(expval.ovr) errgen(49, NULL);
        if(expval.siz==SIZDWD) {
            mchbvst(0x0066);
            mchbvst(opcod+1);
        }
        else if(expval.siz == SIZBYT) mchbvst(opcod);
        else {
            mchbvst(opcod+1);
            if(expval.siz != SIZWRD) errgen(45, NULL);
        }
    }
    return pt;
}

/* JCXZ, LOOP, LOOPE, LOOPNE, LOOPNZ, LOOPZ */
static char *op16(char *pt) {

    int             typ;
    unsigned long lx;

    pt=expr(pt);
    typ=typudl(&expval);
    if(typ=TYPLBL) {
        lx=expval.p2val-lac-2;
        mchbvst(opcod);
        if(lx>=BYTMIN || lx<=BYTMAX) mchbvst((int)lx);
        else {
            errgen(47, NULL);
            mchbvst(0);
        }
    }
    else errgen(46, NULL);
    return pt;
}

/* POP */
static char *op17(char *pt) {

    int     typ,
            reg;

    pt=expr(pt);
    typ=typudv(&expval);
    if(prctyp<=2) {
        if(typ==TYPWRG) mchbvst(0x0058+(expval.bsr & 0x00FF));
        else if(typ==TYPSRG) {
            reg=(expval.bsr&0x00FF)<<3;
            mchbvst(0x0007|reg);
        }
        else if(typ==TYPVAR || typ==TYPIND) {
            mr16(0x008F, 0x0000, &expval);
            if(expval.siz!=SIZWRD) errgen(45, NULL);
        }
        else errgen(46, NULL);
    }
    else {
        if(typ==TYPWRG) mchbvst(0x0058+(expval.bsr & 0x00FF));
        else if(typ==TYPERG) {
            mchbvst(0x0066);
            mchbvst(0x0058+(expval.bsr & 0x00FF));
        }
        else if(typ==TYPSRG) {
            reg=(expval.bsr&0x00FF)<<3;
            mchbvst(0x0007|reg);
        }
        else if(typ==TYPSRX) {
            mchbvst(0x000F);
            reg = expval.bsr & 0x00FF;
            if(reg==4) reg=0x00A1;
            else reg=0x00A9;
            mchbvst(reg);
        }
        else if(typ==TYPVAR || typ==TYPIND || typ==TYPINX) {
            if(expval.siz == SIZDWD) {
                mchbvst(0x0066);
                mr32(0x008F, 0x0000, &expval);
            }
            else {
                mr32(0x008F, 0x0000, &expval);
                if(expval.siz!=SIZWRD) errgen(45, NULL);
            }
        }
        else errgen(46, NULL);
    }
    return pt;
}

/* PUSH */
static char *op18(char *pt) {

    int     typ,
            reg;

    pt=expr(pt);
    typ=typudv(&expval);
    if(prctyp==0) {
        if(typ==TYPWRG) mchbvst(0x0050+(expval.bsr & 0x00FF));
        else if(typ==TYPSRG) {
            reg=(expval.bsr&0x00FF)<<3;
            mchbvst(0x0006|reg);
        }
        else if(typ==TYPVAR || typ==TYPIND) {
            mr16(0x00FF, 0x0030, &expval);
            if(expval.siz!=SIZWRD) errgen(45, NULL);
        }
    }
    else if(prctyp==1 || prctyp==2) {
        if(typ==TYPWRG) mchbvst(0x0050+(expval.bsr & 0x00FF));
        else if(typ==TYPSRG) {
            reg=(expval.bsr & 0x00FF)<<3;
            mchbvst(0x0006 | reg);
        }
        else if(typ==TYPNUM) {
            if(chkis8(&expval)) {
                mchbvst(0x006A);
                mchbvst((int)expval.p2val);
            }
            else {
                mchbvst(0x0068);
                mchwvst((int)expval.p2val);
            }
        }
        else if(typ==TYPVAR || typ==TYPIND) {
            mr16(0x00FF, 0x0030, &expval);
            if(expval.siz!=SIZWRD) errgen(45, NULL);
        }
        else errgen(46, NULL);
    }
    else {
        if(typ==TYPWRG) mchbvst(0x0050+(expval.bsr & 0x00FF));
        else if(typ==TYPERG) {
            mchbvst(0x0066);
            mchbvst(0x0050+(expval.bsr & 0x00FF));
        }
        else if(typ==TYPSRG) {
            reg=(expval.bsr & 0x00FF)<<3;
            mchbvst(0x0006 | reg);
        }
        else if(typ==TYPSRX) {
            mchbvst(0x000F);
```


アセンブラ用語集
汎用レジスタ　AX, BX, CX, DXの４つのレジスタを汎用レジスタという．レジスタ長は16ビット以外に８ビットとしても利用できる．

```
            reg = expval.bsr & 0x00FF;
            if(reg==4) reg=0x00A0;
            else reg=0x00A8;
            mchbvst(reg);
        }
        else if(typ==TYPNUM) {
            if(chkis8(&expval)) {
                mchbvst(0x006A);
                mchbvst((int)expval.p2val);
            }
            else if(chkis16(&expval)) {
                mchbvst(0x0068);
                mchwvst((int)expval.p2val);
            }
            else {
                mchbvst(0x0066);
                mchbvst(0x0068);
                mchlvst(expval.p2val);
            }
        }
        else if(typ==TYPVAR || typ==TYPIND || typ==TYPINX) {
            if(expval.siz == SIZDWD) {
                mchbvst(0x0066);
                mr32(0x00FF, 0x0030, &expval);
            }
            else {
                mr32(0x00FF, 0x0030, &expval);
                if(expval.siz!=SIZWRD) errgen(45, NULL);
            }
        }
        else errgen(46, NULL);
    }
    return pt;
}

/* RET, RETF */
static char *op19(char *pt) {

    int     chk,
            typ;

    chk=isend(pt);
    if(chk==FALSE) {
        pt=expr(pt);
        typ=chktyp(&expval);
        if(typ==TYPNUM) {
            mchbvst(opcod);
            mchwvst((int)expval.p2val);
        }
        else errgen(46, NULL);
    }
    else mchbvst(opcod+1);
    return pt;
}

/* XLAT */
static char *op20(char *pt) {

    int     chk;

    chk=isend(pt);
    if(chk==FALSE) {
        pt=expr(pt);
        expval.ovr=segcnv(expval.ovr);
        if(expval.ovr) mchbvst(expval.ovr);
    }
    mchbvst(opcod);
    return pt;
}

/* LGDT, LIDT, SGDT, SIDT */
static char *op21(char *pt) {

    int     typ,
            reg;

    pt=expr(pt);
    typ=chktyp(&expval);
    if(passno==1 && (expval.att & DEF)==0 && typ==TYPNUM) typ=TYPVAR;
    reg=opcod & 0x00FF;
    opcod=(opcod >> 8) & 0x00FF;
    opcod |= 0x0F00;
    if(prctyp==2) {
        if(reg==0x0038) errgen(44, NULL);
        else if(typ==TYPVAR || typ==TYPIND) {
            mr16(opcod, reg, &expval);
            if(expval.siz!=SIZFWD) errgen(45, NULL);
        }
        else errgen(46, NULL);
    }
    else if(prctyp>=3) {
        if(reg==0x0038 && prctyp==3) errgen(44, NULL);
        else if(typ==TYPVAR || typ==TYPIND || typ==TYPINX) {
            mr32(opcod, reg, &expval);
            if(reg==0x0038) return pt;
            else if(expval.siz!=SIZFWD) errgen(45, NULL);
        }
        else errgen(46, NULL);
    }
    else errgen(42, NULL);
    return pt;
}

/* LLDT, LMSW, LTR, SLDT, SMSW, STR, VERR, VERW */
static char *op22(char *pt) {

    int     typ,
            reg;

    pt=expr(pt);
    typ=typudv(&expval);
    reg=opcod & 0x00FF;
    opcod=(opcod >> 8)&0x00FF;
    opcod|=0x0F00;
    if(prctyp==2) {
        if(typ==TYPWRG) mr16(opcod, reg, &expval);
        else if(typ==TYPVAR || typ==TYPIND) {
            mr16(opcod, reg, &expval);
            if(expval.siz!=SIZWRD) errgen(45, NULL);
        }
        else errgen(46, NULL);
    }
    else if(prctyp>=3) {
        if(typ==TYPWRG) mr32(opcod, reg, &expval);
        else if(typ==TYPERG) {
            mchbvst(0x0066);
            mr32(opcod, reg, &expval);
        }
        else if(typ==TYPVAR || typ==TYPIND || typ==TYPINX) {
            if(expval.siz == SIZDWD) {
                mchbvst(0x0066);
                mr32(opcod, reg, &expval);
            }
            else {
                mr32(opcod, reg, &expval);
                if(expval.siz!=SIZWRD) errgen(45, NULL);
            }
        }
        else errgen(46, NULL);
    }
    return pt;
}

/* SETcc */
static char *op23(char *pt) {

    int     typ;

    pt=expr(pt);
    typ=chktyp(&expval);
    if(passno==1 && (expval.att & DEF)==0 && typ==TYPNUM) typ=TYPVAR;
    if(prctyp>=3) {
        if(typ==TYPBRG) mr32(opcod, 0x0000, &expval);
        else if(typ==TYPVAR || typ==TYPIND || typ==TYPINX) {
            mr32(opcod, 0x0000, &expval);
            if(expval.siz!=SIZBYT) errgen(45, NULL);
        }
        else errgen(46, NULL);
    }
    else errgen(43, NULL);
    return pt;
}

/* BSWAP */
static char *op24(char *pt) {

    int     typ;

    pt=expr(pt);
    typ=chktyp(&expval);
    if(prctyp>=4){
        if(typ==TYPERG) {
            mchbvst(0x0066);
            mchbvst(0x000F);
            mchbvst(0x00C8+(expval.bsr & 0x00FF));
        }
        else errgen(46, NULL);
    }
    else errgen(44, NULL);
    return pt;
}

/* ADC, ADD, CMP, SBB, SUB */
static char *op25(char *pt) {

    if(prctyp<=2) pt=op25a(pt);
    else pt=op25b(pt);
    return pt;
}

static char *op25a(char *pt) {

    int             typ1,
                    typ2;

    pt=expr(pt);
    memcpy(&opd1, &expval, sizeof(struct opdv));
    pt=nxtod(pt);
    pt=expr(pt);
    memcpy(&opd2, &expval, sizeof(struct opdv));
    typ1=typudv(&opd1);
    typ2=typudv(&opd2);
    if(typ1==TYPBRG) {
        if(chkal(&opd1) && typ2==TYPNUM) {
            mchbvst(opcod+4);
            mchbvst((int)opd2.p2val);
        }
        else if(typ2==TYPNUM) {
            mr16(0x0080, opcod, &opd1);
            mchbvst((int)opd2.p2val);
        }
        else if(typ2==TYPBRG) mr16(opcod, (opd2.bsr & 0x00FF)<<3, &opd1);
        else if(typ2==TYPVAR || typ2==TYPIND) {
            mr16(opcod+2, (opd1.bsr & 0x00FF)<<3, &opd2);
            if(opd2.siz!=SIZBYT && opd2.siz!=SIZNON) errgen(50, NULL);
        }
        else errgen(46, NULL);
    }
    else if(typ1==TYPWRG) {
        if(chkax(&opd1) && typ2==TYPNUM) {
            mchbvst(opcod+5);
            mchwvst((int)opd2.p2val);
        }
        else if(typ2==TYPNUM) {
            if(chkis8(&opd2)) {
                mr16(0x0083, opcod, &opd1);
                mchbvst((int)opd2.p2val);
            }
            else {
                mr16(0x0081, opcod, &opd1);
                mchwvst((int)opd2.p2val);
            }
        }
        else if(typ2==TYPWRG)
            mr16(opcod+1, (opd2.bsr & 0x00FF)<<3, &opd1);
        else if(typ2==TYPVAR || typ2==TYPIND) {
            mr16(opcod+3, (opd1.bsr & 0x00FF)<<3, &opd2);
            if(opd2.siz!=SIZWRD && opd2.siz!=SIZNON) errgen(50, NULL);
        }
        else errgen(46, NULL);
    }
    else if(typ1==TYPVAR || typ1==TYPIND) {
        if(typ2==TYPNUM) {
            if(opd1.siz == SIZBYT) {
                mr16(0x0080, opcod, &opd1);
                mchbvst((int)opd2.p2val);
```

```
            }
            else {
                if(chkis8(&opd2)) {
                    mr16(0x0083, opcod, &opd1);
                    mchbvst((int)opd2.p2val);
                }
                else {
                    mr16(0x0081, opcod, &opd1);
                    mchwvst((int)opd2.p2val);
                }
                if(opd1.siz!=SIZWRD) errgen(45, NULL);
            }
        }
        else if(typ2==TYPBRG) {
            mr16(opcod, (opd2.bsr & 0x00FF)<<3, &opd1);
            if(opd1.siz!=SIZBYT && opd1.siz!=SIZNON) errgen(50, NULL);
        }
        else if(typ2==TYPWRG) {
            mr16(opcod+1, (opd2.bsr & 0x00FF)<<3, &opd1);
            if(opd1.siz!=SIZWRD && opd1.siz!=SIZNON) errgen(50, NULL);
        }
        else errgen(46, NULL);
    }
    return pt;
}

static char *op25b(char *pt) {

    int             typ1,
                    typ2;

    pt=expr(pt);
    memcpy(&opd1, &expval, sizeof(struct opdv));
    pt=nxtod(pt);
    pt=expr(pt);
    memcpy(&opd2, &expval, sizeof(struct opdv));
    typ1=typudv(&opd1);
    typ2=typudv(&opd2);
    if(typ1==TYPBRG) {
        if(chkal(&opd1) && typ2==TYPNUM) {
            mchbvst(opcod+4);
            mchbvst((int)opd2.p2val);
        }
        else if(typ2==TYPNUM) {
            mr32(0x0080, opcod, &opd1);
            mchbvst((int)opd2.p2val);
        }
        else if(typ2==TYPBRG) mr32(opcod, (opd2.bsr & 0x00FF)<<3, &opd1);
        else if(typ2==TYPVAR || typ2==TYPIND || typ2==TYPINX) {
            mr32(opcod+2, (opd1.bsr & 0x00FF)<<3, &opd2);
            if(opd2.siz!=SIZBYT && opd2.siz!=SIZNON) errgen(50, NULL);
        }
        else errgen(46, NULL);
    }
    else if(typ1==TYPWRG) {
        if(chkax(&opd1) && typ2==TYPNUM) {
            mchbvst(opcod+5);
            mchwvst((int)opd2.p2val);
        }
        else if(typ2==TYPNUM) {
            if(chkis8(&opd2)) {
                mr32(0x0083, opcod, &opd1);
                mchbvst((int)opd2.p2val);
            }
            else {
                mr32(0x0081, opcod, &opd1);
                mchwvst((int)opd2.p2val);
            }
        }
        else if(typ2==TYPWRG)
            mr32(opcod+1, (opd2.bsr & 0x00FF)<<3, &opd1);
        else if(typ2==TYPVAR || typ2==TYPIND || typ2==TYPINX) {
            mr32(opcod+3, (opd1.bsr & 0x00FF)<<3, &opd2);
            if(opd2.siz!=SIZWRD && opd2.siz!=SIZNON) errgen(50, NULL);
        }
        else errgen(46, NULL);
    }
    else if(typ1==TYPERG) {
        if(chkeax(&opd1) && typ2==TYPNUM) {
            mchbvst(0x0066);
            mchbvst(opcod+5);
            mchlvst(opd2.p2val);
        }
        else if(typ2==TYPNUM) {
            if(chkis8(&opd2)) {
                mchbvst(0x0066);
                mr32(0x0083, opcod, &opd1);
                mchbvst((int)opd2.p2val);
            }
            else {
                mchbvst(0x0066);
                mr32(0x0081, opcod, &opd1);
                mchlvst(opd2.p2val);
            }
        }
        else if(typ2==TYPERG) {
            mchbvst(0x0066);
            mr32(opcod+1, (opd2.bsr & 0x00FF)<<3, &opd1);
        }
        else if(typ2==TYPVAR || typ2==TYPIND || typ2==TYPINX) {
            mchbvst(0x0066);
            mr32(opcod+3, (opd1.bsr & 0x00FF)<<3, &opd2);
            if(opd2.siz!=SIZDWD && opd2.siz!=SIZNON) errgen(50, NULL);
        }
        else errgen(46, NULL);
    }
    else if(typ1==TYPVAR || typ1==TYPIND || typ1==TYPINX) {
        if(typ2==TYPNUM) {
            if(opd1.siz == SIZBYT) {
                mr32(0x0080, opcod, &opd1);
                mchbvst((int)opd2.p2val);
            }
            else if(opd1.siz == SIZDWD) {
                mchbvst(0x0066);
                if(chkis8(&opd2)) {
                    mr32(0x0083, opcod, &opd1);
                    mchbvst((int)opd2.p2val);
                }
                else {
                    mr32(0x0081, opcod, &opd1);
                    mchlvst(opd2.p2val);
                }
            }
            else {
                if(chkis8(&opd2)) {
                    mr32(0x0083, opcod, &opd1);
                    mchbvst((int)opd2.p2val);
                }
                else {
                    mr32(0x0081, opcod, &opd1);
                    mchwvst((int)opd2.p2val);
                }
                if(opd1.siz!=SIZWRD) errgen(45, NULL);
            }
        }
        else if(typ2==TYPBRG) {
            mr32(opcod, (opd2.bsr & 0x00FF)<<3, &opd1);
            if(opd1.siz!=SIZBYT && opd1.siz!=SIZNON) errgen(50, NULL);
        }
        else if(typ2==TYPWRG) {
            mr32(opcod+1, (opd2.bsr & 0x00FF)<<3, &opd1);
            if(opd1.siz!=SIZWRD && opd1.siz!=SIZNON) errgen(50, NULL);
        }
        else if(typ2==TYPERG) {
            mchbvst(0x0066);
            mr32(opcod+1, (opd2.bsr & 0x00FF)<<3, &opd1);
            if(opd1.siz!=SIZDWD && opd1.siz!=SIZNON) errgen(50, NULL);
        }
        else errgen(46, NULL);
    }
    return pt;
}

/* AND, OR, XOR */
static char *op26(char *pt) {

    if(prctyp<=2) pt=op26a(pt);
    else pt=op26b(pt);
    return pt;
}

static char *op26a(char *pt) {

    int             typ1,
                    typ2;

    pt=expr(pt);
    memcpy(&opd1, &expval, sizeof(struct opdv));
    pt=nxtod(pt);
    pt=expr(pt);
    memcpy(&opd2, &expval, sizeof(struct opdv));
    typ1=typudv(&opd1);
    typ2=typudv(&opd2);
    if(typ1==TYPBRG) {
        if(chkal(&opd1) && typ2==TYPNUM) {
            mchbvst(opcod+4);
            mchbvst((int)opd2.p2val);
        }
        else if(typ2==TYPNUM) {
            mr16(0x0080, opcod, &opd1);
            mchbvst((int)opd2.p2val);
        }
        else if(typ2==TYPBRG) mr16(opcod, (opd2.bsr & 0x00FF)<<3, &opd1);
        else if(typ2==TYPVAR || typ2==TYPIND) {
            mr16(opcod+2, (opd1.bsr & 0x00FF)<<3, &opd2);
            if(opd2.siz!=SIZBYT && opd2.siz!=SIZNON) errgen(50, NULL);
        }
        else errgen(46, NULL);
    }
    else if(typ1==TYPWRG) {
        if(chkax(&opd1) && typ2==TYPNUM) {
            mchbvst(opcod+5);
            mchwvst((int)opd2.p2val);
        }
        else if(typ2==TYPNUM) {
            mr16(0x0081, opcod, &opd1);
            mchwvst((int)opd2.p2val);
        }
        else if(typ2==TYPWRG)
            mr16(opcod+1, (opd2.bsr & 0x00FF)<<3, &opd1);
        else if(typ2==TYPVAR || typ2==TYPIND) {
            mr16(opcod+3, (opd1.bsr & 0x00FF)<<3, &opd2);
            if(opd2.siz!=SIZWRD && opd2.siz!=SIZNON) errgen(50, NULL);
        }
        else errgen(46, NULL);
    }
    else if(typ1==TYPVAR || typ1==TYPIND) {
        if(typ2==TYPNUM) {
            if(opd1.siz == SIZBYT) {
                mr16(0x0080, opcod, &opd1);
                mchbvst((int)opd2.p2val);
            }
            else {
                mr16(0x0081, opcod, &opd1);
                mchwvst((int)opd2.p2val);
                if(opd1.siz!=SIZWRD) errgen(45, NULL);
            }
        }
        else if(typ2==TYPBRG) {
            mr16(opcod, (opd2.bsr & 0x00FF)<<3, &opd1);
            if(opd1.siz!=SIZBYT && opd1.siz!=SIZNON) errgen(50, NULL);
        }
        else if(typ2==TYPWRG) {
            mr16(opcod+1, (opd2.bsr & 0x00FF)<<3, &opd1);
            if(opd1.siz!=SIZWRD && opd1.siz!=SIZNON) errgen(50, NULL);
        }
        else errgen(46, NULL);
    }
    return pt;
}

static char *op26b(char *pt) {

    int             typ1,
                    typ2;

    pt=expr(pt);
    memcpy(&opd1, &expval, sizeof(struct opdv));
    pt=nxtod(pt);
    pt=expr(pt);
    memcpy(&opd2, &expval, sizeof(struct opdv));
    typ1=typudv(&opd1);
    typ2=typudv(&opd2);
    if(typ1==TYPBRG) {
        if(chkal(&opd1) && typ2==TYPNUM) {
            mchbvst(opcod+4);
```

アセンブラ用語集

ファンクションコール　MS-DOSシステムコールのひとつで，INT 21Hを指す．AHレジスタに機能（ファンクション）番号を入れ呼び出すことで多くの機能を実現する．

```c
            mchbvst((int)opd2.p2val);
        }
        else if(typ2==TYPNUM) {
            mr32(0x0080, opcod, &opd1);
            mchbvst((int)opd2.p2val);
        }
        else if(typ2==TYPBRG) mr32(opcod, (opd2.bsr & 0x00FF)<<3, &opd1);
        else if(typ2==TYPVAR || typ2==TYPIND || typ2==TYPINX) {
            mr32(opcod+2, (opd1.bsr & 0x00FF)<<3, &opd2);
            if(opd2.siz!=SIZBYT && opd2.siz!=SIZNON) errgen(50, NULL);
        }
        else errgen(46, NULL);
    }
    else if(typ1==TYPWRG) {
        if(chkax(&opd1) && typ2==TYPNUM) {
            mchbvst(opcod+5);
            mchwvst((int)opd2.p2val);
        }
        else if(typ2==TYPNUM) {
            mr32(0x0081, opcod, &opd1);
            mchwvst((int)opd2.p2val);
        }
        else if(typ2==TYPWRG)
            mr32(opcod+1, (opd2.bsr & 0x00FF)<<3, &opd1);
        else if(typ2==TYPVAR || typ2==TYPIND || typ2==TYPINX) {
            mr32(opcod+3, (opd1.bsr & 0x00FF)<<3, &opd2);
            if(opd2.siz!=SIZWRD && opd2.siz!=SIZNON) errgen(50, NULL);
        }
        else errgen(46, NULL);
    }
    else if(typ1==TYPERG) {
        if(chkeax(&opd1) && typ2==TYPNUM) {
            mchbvst(0x0066);
            mchbvst(opcod+5);
            mchlvst(opd2.p2val);
        }
        else if(typ2==TYPNUM) {
            mchbvst(0x0066);
            mr32(0x0081, opcod, &opd1);
            mchlvst(opd2.p2val);
        }
        else if(typ2==TYPERG) {
            mchbvst(0x0066);
            mr32(opcod+1, (opd2.bsr & 0x00FF)<<3, &opd1);
        }
        else if(typ2==TYPVAR || typ2==TYPIND || typ2==TYPINX) {
            mchbvst(0x0066);
            mr32(opcod+3, (opd1.bsr & 0x00FF)<<3, &opd2);
            if(opd2.siz!=SIZDWD && opd2.siz!=SIZNON) errgen(50, NULL);
        }
        else errgen(46, NULL);
    }
    else if(typ1==TYPVAR || typ1==TYPIND || typ1==TYPINX) {
        if(typ2==TYPNUM) {
            if(opd1.siz == SIZBYT) {
                mr32(0x0080, opcod, &opd1);
                mchbvst((int)opd2.p2val);
            }
            else if(opd1.siz == SIZDWD) {
                mchbvst(0x0066);
                mr32(0x0081, opcod, &opd1);
                mchlvst(opd2.p2val);
            }
            else {
                mr32(0x0081, opcod, &opd1);
                mchwvst((int)opd2.p2val);
                if(opd1.siz!=SIZWRD) errgen(45, NULL);
            }
        }
        else if(typ2==TYPBRG) {
            mr32(opcod, (opd2.bsr & 0x00FF)<<3, &opd1);
            if(opd1.siz!=SIZBYT && opd1.siz!=SIZNON) errgen(50, NULL);
        }
        else if(typ2==TYPWRG) {
            mr32(opcod+1, (opd2.bsr & 0x00FF)<<3, &opd1);
            if(opd1.siz!=SIZWRD && opd1.siz!=SIZNON) errgen(50, NULL);
        }
        else if(typ2==TYPERG) {
            mchbvst(0x0066);
            mr32(opcod+1, (opd2.bsr & 0x00FF)<<3, &opd1);
            if(opd1.siz!=SIZDWD && opd1.siz!=SIZNON) errgen(50, NULL);
        }
        else errgen(46, NULL);
    }
    return pt;
}

/* CMPS, MOVS */
static char *op27(char *pt) {

    pt=expr(pt);
    memcpy(&opd1, &expval, sizeof(struct opdv));
    pt=nxtod(pt);
    pt=expr(pt);
    memcpy(&opd2, &expval, sizeof(struct opdv));
    opd2.ovr=segcnv(opd2.ovr);
    if(prctyp<=2) {
        if(opd2.ovr) mchbvst(opd2.ovr);
        if(opd1.siz != opd2.siz) errgen(50, NULL);
        if(opd1.siz == SIZBYT) mchbvst(opcod);
        else {
            mchbvst(opcod+1);
            if(opd1.siz!=SIZWRD) errgen(45, NULL);
        }
    }
    else {
        if(opd2.ovr) mchbvst(opd2.ovr);
        if(opd1.siz != opd2.siz) errgen(50, NULL);
        if(opd1.siz == SIZBYT) mchbvst(opcod);
        else if(opd1.siz == SIZDWD) {
            mchbvst(0x0066);
            mchbvst(opcod+1);
        }
        else {
            mchbvst(opcod+1);
            if(opd1.siz!=SIZWRD) errgen(45, NULL);
        }
    }
    return pt;
}

/* ESC */
static char *op28(char *pt) {

    int         typ1,
                typ2,
                rg1,
                rg2;

    pt=expr(pt);
    memcpy(&opd1, &expval, sizeof(struct opdv));
    pt=nxtod(pt);
    pt=expr(pt);
    memcpy(&opd2, &expval, sizeof(struct opdv));
    typ1=chktyp(&opd1);
    typ2=typudv(&opd2);
    if(typ1==TYPNUM) {
        rg1=(int)opd1.p2val;
        rg2=(rg1 & 0x0007)<<3;
        rg1=(rg1 & 0x0038)>>3;
        if(prctyp<=2) {
            if(typ2==TYPWRG) mr16(0x00D8+rg1, rg2, &opd2);
            else if(typ2==TYPVAR || typ2==TYPIND) {
                mr16(0x00D8+rg1, rg2, &opd2);
                if(opd2.siz!=SIZWRD && opd2.siz!=SIZNON)
                    errgen(45, NULL);
            }
            else errgen(46, NULL);
        }
        else {
            if(typ2==TYPWRG) mr32(0x00D8+rg1, rg2, &opd2);
            else if(typ2==TYPERG) {
                mchbvst(0x0066);
                mr32(0x00D8+rg1, rg2, &opd2);
            }
            else if(typ2==TYPVAR || typ2==TYPIND ||typ2==TYPINX) {
                if(opd2.siz == SIZDWD) {
                    mchbvst(0x0066);
                    mr32(0x00D8+rg1, rg2, &opd2);
                }
                else {
                    mr32(0x00D8+rg1, rg2, &opd2);
                    if(opd2.siz!=SIZWRD && opd2.siz!=SIZNON)
                        errgen(45, NULL);
                }
            }
            else errgen(46, NULL);
        }
    }
    else errgen(46, NULL);
    return pt;
}

/* IN */
static char *op29(char *pt) {

    int         typ;

    pt=expr(pt);
    memcpy(&opd1, &expval, sizeof(struct opdv));
    pt=nxtod(pt);
    pt=expr(pt);
    memcpy(&opd2, &expval, sizeof(struct opdv));
    typ=chktyp(&opd2);
    if(prctyp<=2) {
        if(chkal(&opd1)) {
            if(chkdx(&opd2)) mchbvst(0x00EC);
            else if(typ==TYPNUM) {
                mchbvst(0x00E4);
                mchbvst((int)opd2.p2val);
            }
            else errgen(46, NULL);
        }
        else if(chkax(&opd1)) {
            if(chkdx(&opd2)) mchbvst(0x00ED);
            else if(typ==TYPNUM) {
                mchbvst(0x00E5);
                mchbvst((int)opd2.p2val);
            }
            else errgen(46, NULL);
        }
        else errgen(46, NULL);
    }
    else {
        if(chkal(&opd1)) {
            if(chkdx(&opd2)) mchbvst(0x00EC);
            else if(typ==TYPNUM) {
                mchbvst(0x00E4);
                mchbvst((int)opd2.p2val);
            }
            else errgen(46, NULL);
        }
        else if(chkax(&opd1)) {
            if(chkdx(&opd2)) mchbvst(0x00ED);
            else if(typ==TYPNUM) {
                mchbvst(0x00E5);
                mchbvst((int)opd2.p2val);
            }
            else errgen(46, NULL);
        }
        else if(chkeax(&opd1)) {
            if(chkdx(&opd2)) {
                mchbvst(0x0066);
                mchbvst(0x00EC);
            }
            else if(typ==TYPNUM) {
                mchbvst(0x0066);
                mchbvst(0x00E5);
                mchbvst((int)opd2.p2val);
            }
            else errgen(46, NULL);
        }
        else errgen(46, NULL);
    }
    return pt;
}

/* LDS, LES */
static char *op30(char *pt) {

    int         typ1,
                typ2;

    pt=expr(pt);
    memcpy(&opd1, &expval, sizeof(struct opdv));
    pt=nxtod(pt);
```

アセンブラ用語集
フラグレジスタ　演算命令の実行結果などを1ビット単位で格納する特別なレジスタ.

```
	pt=expr(pt);
	memcpy(&opd2, &expval, sizeof(struct opdv));
	typ1=chktyp(&opd1);
	typ2=typudf(&opd2);
	if(prctyp<=2) {
		if(typ2==TYPVAR || typ2==TYPIND) {
			if(typ1==TYPWRG) {
				mr16(opcod, (opd1.bsr & 0x00FF)<<3, &opd2);
				if(opd2.siz!=SIZDWD && opd2.siz!=SIZNON) errgen(45, NULL);
			}
			else errgen(46, NULL);
		}
		else errgen(46, NULL);
	}
	else {
		if(typ2==TYPVAR || typ2==TYPIND || typ2==TYPINX) {
			if(typ1==TYPWRG) {
				mr32(opcod, (opd1.bsr & 0x00FF)<<3, &opd2);
				if(opd2.siz!=SIZDWD && opd2.siz!=SIZNON)
					errgen(45, NULL);
			}
			else if(typ1==TYPERG) {
				mchbvst(0x0066);
				mr32(opcod, (opd1.bsr & 0x00FF)<<3, &opd2);
				if(opd2.siz!=SIZFWD && opd2.siz!=SIZNON)
					errgen(45, NULL);
			}
			else errgen(46, NULL);
		}
		else errgen(46, NULL);
	}
	return pt;
}

/* LEA */
static char *op31(char *pt) {

	int		typ1,
			typ2;

	pt=expr(pt);
	memcpy(&opd1, &expval, sizeof(struct opdv));
	pt=nxtod(pt);
	pt=expr(pt);
	memcpy(&opd2, &expval, sizeof(struct opdv));
	typ1=chktyp(&opd1);
	typ2=chktyp(&opd2);
	if(passno==1 && (opd2.att & DEF)==0 && typ2==TYPNUM) {
		typ2=TYPVAR;
		if(prctyp<=2) opd2.siz=SIZDWD;
		else opd2.siz=SIZFWD;
	}
	if(prctyp<=2) {
		if(typ2==TYPVAR || typ2==TYPIND || typ2==TYPLBL) {
			if(typ1==TYPWRG) {
				mr16(opcod, (opd1.bsr & 0x00FF)<<3, &opd2);
				if(opd2.siz!=SIZWRD && opd2.siz!=SIZNON)
					errgen(45, NULL);
			}
			else errgen(46, NULL);
		}
		else errgen(46, NULL);
	}
	else {
		if(typ2==TYPVAR || typ2==TYPIND || typ2==TYPINX ||
			typ2==TYPLBL) {
			if(typ1==TYPWRG) {
				mr32(opcod, (opd1.bsr & 0x00FF)<<3, &opd2);
				if(opd2.siz!=SIZWRD && opd2.siz!=SIZNON)
					errgen(45, NULL);
			}
			else if(typ1==TYPERG) {
				mchbvst(0x0066);
				mr32(opcod, (opd1.bsr & 0x00FF)<<3, &opd2);
				if(opd2.siz!=SIZDWD && opd2.siz!=SIZNON)
					errgen(45, NULL);
			}
			else errgen(46, NULL);
		}
		else errgen(46, NULL);
	}
	return pt;
}

/* MOV */
static char *op32(char *pt) {

	if(prctyp<=2) pt=op32a(pt);
	else pt=op32b(pt);
	return pt;
}

static char *op32a(char *pt) {

	int	typ1,
		typ2;

	pt=expr(pt);
	memcpy(&opd1, &expval, sizeof(struct opdv));
	pt=nxtod(pt);
	pt=expr(pt);
	memcpy(&opd2, &expval, sizeof(struct opdv));
	typ1=typudv(&opd1);
	typ2=typudv(&opd2);
	if(chkal(&opd1) && typ2==TYPVAR) {
		m16(0x00A0, &opd2);
		if(opd2.siz!=SIZBYT && opd2.siz!=SIZNON) errgen(50, NULL);
	}
	else if(chkax(&opd1) && typ2==TYPVAR) {
		m16(0x00A1, &opd2);
		if(opd2.siz!=SIZWRD && opd2.siz!=SIZNON) errgen(50, NULL);
	}
	else if(typ1==TYPVAR && chkal(&opd2)) {
		m16(0x00A2, &opd1);
		if(opd1.siz!=SIZBYT && opd1.siz!=SIZNON) errgen(50, NULL);
	}
	else if(typ1==TYPVAR && chkax(&opd2)) {
		m16(0x00A3, &opd1);
		if(opd1.siz!=SIZWRD && opd1.siz!=SIZNON) errgen(50, NULL);
	}
	else if(typ1==TYPBRG) {
		if(typ2==TYPNUM) {
			mchbvst(0x00B0+(opd1.bsr & 0x00FF));
			mchbvst((int)opd2.p2val);
		}
		else if(typ2==TYPBRG)
			mr16(0x0088, (opd2.bsr & 0x00FF)<<3, &opd1);
		else if(typ2==TYPVAR || typ2==TYPIND) {
			mr16(0x008A, (opd1.bsr & 0x00FF)<<3, &opd2);
			if(opd2.siz!=SIZBYT && opd2.siz!=SIZNON) errgen(50, NULL);
		}
		else errgen(46, NULL);
	}
	else if(typ1==TYPWRG) {
		if(typ2==TYPNUM) {
			mchbvst(0x00B8+(opd1.bsr & 0x00FF));
			mchwvst((int)opd2.p2val);
		}
		else if(typ2==TYPWRG)
			mr16(0x0089, (opd2.bsr & 0x00FF)<<3, &opd1);
		else if(typ2==TYPSRG)
			mr16(0x008C, (opd2.bsr & 0x00FF)<<3, &opd1);
		else if(typ2==TYPVAR || typ2==TYPIND) {
			mr16(0x008B, (opd1.bsr & 0x00FF)<<3, &opd2);
			if(opd2.siz!=SIZWRD && opd2.siz!=SIZNON) errgen(50, NULL);
		}
		else errgen(46, NULL);
	}
	else if(typ1==TYPSRG) {
		if(typ2==TYPWRG) mr16(0x008E, (opd1.bsr & 0x00FF)<<3, &opd2);
		else if(typ2==TYPVAR || typ2==TYPIND) {
			mr16(0x008E, (opd1.bsr & 0x00FF)<<3, &opd2);
			if(opd2.siz!=SIZWRD && opd2.siz!=SIZNON) errgen(50, NULL);
		}
		else errgen(46, NULL);
	}
	else if(typ1==TYPVAR || typ1==TYPIND) {
		if(typ2==TYPBRG) {
			mr16(0x0088, (opd2.bsr & 0x00FF)<<3, &opd1);
			if(opd1.siz!=SIZBYT && opd1.siz!=SIZNON) errgen(50, NULL);
		}
		else if(typ2==TYPWRG) {
			mr16(0x0089, (opd2.bsr & 0x00FF)<<3, &opd1);
			if(opd1.siz!=SIZWRD && opd1.siz!=SIZNON) errgen(50, NULL);
		}
		else if(typ2==TYPSRG) {
			mr16(0x008C, (opd2.bsr & 0x00FF)<<3, &opd1);
			if(opd1.siz!=SIZWRD && opd1.siz!=SIZNON) errgen(50, NULL);
		}
		else if(typ2==TYPNUM) {
			if(opd1.siz == SIZBYT) {
				mr16(0x00C6, 0x0000, &opd1);
				mchbvst((int)opd2.p2val);
			}
			else {
				mr16(0x00C7, 0x0000, &opd1);
				mchwvst((int)opd2.p2val);
				if(opd1.siz!=SIZWRD) errgen(45, NULL);
			}
		}
		else errgen(46, NULL);
	}
	else errgen(46, NULL);
	return pt;
}

#pragma optimize("le", off)
static char *op32b(char *pt) {

	int	typ1,
		typ2,
		reg;

	pt=expr(pt);
	memcpy(&opd1, &expval, sizeof(struct opdv));
	pt=nxtod(pt);
	pt=expr(pt);
	memcpy(&opd2, &expval, sizeof(struct opdv));
	typ1=typudv(&opd1);
	typ2=typudv(&opd2);
	if(chkal(&opd1) && typ2==TYPVAR) {
		m16(0x00A0, &opd2);
		if(opd2.siz!=SIZBYT && opd2.siz!=SIZNON) errgen(50, NULL);
	}
	else if(chkeax(&opd1) && typ2==TYPVAR) {
		mchbvst(0x0066);
		m16(0x00A1, &opd2);
		if(opd2.siz!=SIZDWD && opd2.siz!=SIZNON) errgen(50, NULL);
	}
	else if(chkax(&opd1) && typ2==TYPVAR) {
		m16(0x00A1, &opd2);
		if(opd2.siz!=SIZWRD && opd2.siz!=SIZNON) errgen(50, NULL);
	}
	else if(typ1==TYPVAR && chkal(&opd2)) {
		m16(0x00A2, &opd1);
		if(opd1.siz!=SIZBYT && opd1.siz!=SIZNON) errgen(50, NULL);
	}
	else if(typ1==TYPVAR && chkeax(&opd2)) {
		mchbvst(0x0066);
		m16(0x00A3, &opd1);
		if(opd1.siz!=SIZDWD && opd1.siz!=SIZNON) errgen(50, NULL);
	}
	else if(typ1==TYPVAR && chkax(&opd2)) {
		m16(0x00A3, &opd1);
		if(opd1.siz!=SIZWRD && opd1.siz!=SIZNON) errgen(50, NULL);
	}
	else if(typ1==TYPBRG) {
		if(typ2==TYPNUM) {
			mchbvst(0x00B0+(opd1.bsr & 0x00FF));
			mchbvst((int)opd2.p2val);
		}
		else if(typ2==TYPBRG)
			mr32(0x0088, (opd2.bsr & 0x00FF)<<3, &opd1);
		else if(typ2==TYPVAR || typ2==TYPIND || typ2==TYPINX) {
			mr32(0x008A, (opd1.bsr & 0x00FF)<<3, &opd2);
			if(opd2.siz!=SIZBYT && opd2.siz!=SIZNON) errgen(50, NULL);
		}
		else errgen(46, NULL);
	}
	else if(typ1==TYPERG) {
		if(typ2==TYPNUM) {
			mchbvst(0x0066);
			mchbvst(0x00B8+(opd1.bsr & 0x00FF));
			mchlvst(opd2.p2val);
		}
		else if(typ2==TYPWRG) {
```

アセンブラ用語集

プレフィックス　8086の命令の一種で直後の命令を修飾する．ストリング操作命令やリピートプレフィックスなどがある．

```
            mchbvst(0x0066);
            mr32(0x0089, (opd2.bsr & 0x00FF)<<3, &opd1);
        }
        else if(typ2==TYPSRG || typ2==TYPSRX) {
            mchbvst(0x0066);
            mr32(0x008C, (opd2.bsr & 0x00FF)<<3, &opd1);
        }
        else if(typ2==TYPVAR || typ2==TYPIND || typ2==TYPINX) {
            mchbvst(0x0066);
            mr32(0x008B, (opd1.bsr & 0x00FF)<<3, &opd2);
            if(opd2.siz!=SIZDWD && opd2.siz!=SIZNON) errgen(50, NULL);
        }
        else if(typ2==TYPXRG) {
            typ2 = opd2.bsr & 0xFF00;
            if(typ2 == CREG) opcod=0x0F20;
            else if(typ2 == DREG) opcod=0x0F21;
            else opcod=0x0F24;
            reg = (opd2.bsr & 0x00FF)<<3;
            mr32(opcod, reg, &opd1);
        }
        else errgen(46, NULL);
    }
    else if(typ1==TYPWRG) {
        if(typ2==TYPNUM) {
            mchbvst(0x00B8+(opd1.bsr & 0x00FF));
            mchwvst((int)opd2.p2val);
        }
        else if(typ2==TYPWRG)
            mr32(0x0089, (opd2.bsr & 0x00FF)<<3, &opd1);
        else if(typ2==TYPSRG || typ2==TYPSRX)
            mr32(0x008C, (opd2.bsr & 0x00FF)<<3, &opd1);
        else if(typ2==TYPVAR || typ2==TYPIND || typ2==TYPINX) {
            mr32(0x008B, (opd1.bsr & 0x00FF)<<3, &opd2);
            if(opd2.siz!=SIZWRD && opd2.siz!=SIZNON) errgen(50, NULL);
        }
        else errgen(46, NULL);
    }
    else if(typ1==TYPSRG || typ1==TYPSRX) {
        if(typ2==TYPWRG) mr32(0x008E, (opd1.bsr & 0x00FF)<<3, &opd2);
        else if(typ2==TYPERG) {
            mchbvst(0x0066);
            mr32(0x008E, (opd1.bsr & 0x00FF)<<3, &opd2);
        }
        else if(typ2==TYPVAR || typ2==TYPIND || typ2==TYPINX) {
            if(opd2.siz == SIZDWD) {
                mchbvst(0x0066);
                mr32(0x008E, (opd1.bsr & 0x00FF)<<3, &opd2);
            }
            else {
                mr32(0x008E, (opd1.bsr & 0x00FF)<<3, &opd2);
                if(opd2.siz!=SIZWRD && opd2.siz!=SIZNON)
                    errgen(50, NULL);
            }
        }
        else errgen(46, NULL);
    }
    else if(typ1==TYPXRG) {
        if(typ2==TYPERG) {
            typ1 = opd1.bsr & 0xFF00;
            if(typ1 == CREG) opcod=0x0F22;
            else if(typ1 == DREG) opcod=0x0F23;
            else opcod=0x0F26;
            reg = (opd1.bsr & 0x00FF)<<3;
            mr32(opcod, reg, &opd2);
        }
        else errgen(46, NULL);
    }
    else if(typ1==TYPVAR || typ1==TYPIND || typ1==TYPINX) {
        if(typ2==TYPBRG) {
            mr32(0x0088, (opd2.bsr & 0x00FF)<<3, &opd1);
            if(opd1.siz!=SIZBYT && opd1.siz!=SIZNON) errgen(50, NULL);
        }
        else if(typ2==TYPWRG) {
            mr32(0x0089, (opd2.bsr & 0x00FF)<<3, &opd1);
            if(opd1.siz!=SIZWRD && opd1.siz!=SIZNON) errgen(50, NULL);
        }
        else if(typ2==TYPSRG || typ2==TYPSRX) {
            if(opd1.siz==SIZDWD) {
                mchbvst(0x0066);
                mr32(0x008C, (opd2.bsr & 0x00FF)<<3, &opd1);
            }
            else {
                mr32(0x008C, (opd2.bsr & 0x00FF)<<3, &opd1);
                if(opd1.siz!=SIZWRD && opd1.siz!=SIZNON)
                    errgen(50, NULL);
            }
        }
        else if(typ2==TYPERG) {
            mchbvst(0x0066);
            mr32(0x0089, (opd2.bsr & 0x00FF)<<3, &opd1);
            if(opd1.siz!=SIZDWD && opd1.siz!=SIZNON) errgen(50, NULL);
        }
        else if(typ2==TYPNUM) {
            if(opd1.siz == SIZBYT) {
                mr32(0x00C6, 0x0000, &opd1);
                mchbvst((int)opd2.p2val);
            }
            else if(opd1.siz == SIZDWD) {
                mchbvst(0x0066);
                mr32(0x00C7, 0x0000, &opd1);
                mchlvst(opd2.p2val);
            }
            else {
                mr32(0x00C7, 0x0000, &opd1);
                mchwvst((int)opd2.p2val);
                if(opd1.siz!=SIZWRD) errgen(45, NULL);
            }
        }
        else errgen(46, NULL);
    }
    else errgen(46, NULL);
    return pt;
}
#pragma optimize("le", on)

/* OUT */
static char *op33(char *pt) {

    int         typ;

    pt=expr(pt);
    memcpy(&opd1, &expval, sizeof(struct opdv));
    pt=nxtod(pt);
    pt=expr(pt);
    memcpy(&opd2, &expval, sizeof(struct opdv));
    typ=chktyp(&opd1);
    if(prctyp<=2) {
        if(chkal(&opd2)) {
            if(chkdx(&opd1)) mchbvst(0x00EE);
            else if(typ==TYPNUM) {
                mchbvst(0x00E6);
                mchbvst((int)opd1.p2val);
            }
            else errgen(46, NULL);
        }
        else if(chkax(&opd2)) {
            if(chkdx(&opd1)) mchbvst(0x00EF);
            else if(typ==TYPNUM) {
                mchbvst(0x00E7);
                mchbvst((int)opd1.p2val);
            }
            else errgen(46, NULL);
        }
        else errgen(46, NULL);
    }
    else {
        if(chkal(&opd2)) {
            if(chkdx(&opd1)) mchbvst(0x00EE);
            else if(typ==TYPNUM) {
                mchbvst(0x00E6);
                mchbvst((int)opd1.p2val);
            }
            else errgen(46, NULL);
        }
        else if(chkax(&opd2)) {
            if(chkdx(&opd1)) mchbvst(0x00EF);
            else if(typ==TYPNUM) {
                mchbvst(0x00E7);
                mchbvst((int)opd1.p2val);
            }
            else errgen(46, NULL);
        }
        else if(chkeax(&opd2)) {
            if(chkdx(&opd1)) {
                mchbvst(0x0066);
                mchbvst(0x00EF);
            }
            else if(typ==TYPNUM) {
                mchbvst(0x0066);
                mchbvst(0x00E7);
                mchbvst((int)opd1.p2val);
            }
            else errgen(46, NULL);
        }
        else errgen(46, NULL);
    }
    return pt;
}

/* RCL, RCR, ROL, ROR, SAL, SAR, SHL, SHR */
static char *op34(char *pt) {

    pt=expr(pt);
    memcpy(&opd1, &expval, sizeof(struct opdv));
    pt=nxtod(pt);
    pt=expr(pt);
    memcpy(&opd2, &expval, sizeof(struct opdv));
    if(prctyp==0) op34a();
    else if(prctyp==1 || prctyp==2) op34b();
    else op34c();
    return pt;
}

static void op34a(void) {

    int         typ1,
                typ2;

    typ1=typudv(&opd1);
    typ2=chktyp(&opd2);
    if(typ1==TYPBRG) {
        if(chkcl(&opd2)) mr16(0x00D2, opcod, &opd1);
        else if(typ2==TYPNUM) {
            mr16(0x00D0, opcod, &opd1);
            if(opd2.p2val != 1L) errgen(47, NULL);
        }
        else errgen(46, NULL);
    }
    else if(typ1==TYPWRG) {
        if(chkcl(&opd2)) mr16(0x00D3, opcod, &opd1);
        else if(typ2==TYPNUM) {
            mr16(0x00D1, opcod, &opd1);
            if(opd2.p2val != 1L) errgen(47, NULL);
        }
        else errgen(46, NULL);
    }
    else if(typ1==TYPVAR || typ1==TYPIND) {
        if(chkcl(&opd2)) {
            if(opd1.siz == SIZBYT) mr16(0x00D2, opcod, &opd1);
            else {
                mr16(0x00D3, opcod, &opd1);
                if(opd1.siz!=SIZWRD) errgen(45, NULL);
            }
        }
        else if(typ2==TYPNUM) {
            if(opd1.siz == SIZBYT) mr16(0x00D0, opcod, &opd1);
            else {
                mr16(0x00D1, opcod, &opd1);
                if(opd1.siz!=SIZWRD) errgen(45, NULL);
            }
            if(opd2.p2val != 1L) errgen(47, NULL);
        }
        else errgen(46, NULL);
    }
    else errgen(46, NULL);
}

static void op34b(void) {

    int         typ1,
                typ2;

    typ1=typudv(&opd1);
    typ2=chktyp(&opd2);
    if(typ1==TYPBRG) {
        if(chkcl(&opd2)) mr16(0x00D2, opcod, &opd1);
```

```
        else if(typ2==TYPNUM) {
            if(opd2.p2val == 1UL) mr16(0x00D0, opcod, &opd1);
            else {
                mr16(0x00C0, opcod, &opd1);
                mchbvst((int)opd2.p2val);
            }
        }
        else errgen(46, NULL);
    }
    else if(typ1==TYPWRG) {
        if(chkcl(&opd2)) mr16(0x00D3, opcod, &opd1);
        else if(typ2==TYPNUM) {
            if(opd2.p2val == 1UL) mr16(0x00D1, opcod, &opd1);
            else {
                mr16(0x00C1, opcod, &opd1);
                mchbvst((int)opd2.p2val);
            }
        }
        else errgen(46, NULL);
    }
    else if(typ1==TYPVAR || typ1==TYPIND) {
        if(chkcl(&opd2)) {
            if(opd1.siz == SIZBYT) mr16(0x00D2, opcod, &opd1);
            else {
                mr16(0x00D3, opcod, &opd1);
                if(opd1.siz!=SIZWRD) errgen(45, NULL);
            }
        }
        else if(typ2==TYPNUM && opd2.p2val==1UL) {
            if(opd1.siz == SIZBYT) mr16(0x00D0, opcod, &opd1);
            else {
                mr16(0x00D1, opcod, &opd1);
                if(opd1.siz!=SIZWRD) errgen(45, NULL);
            }
        }
        else if(typ2==TYPNUM) {
            if(opd1.siz == SIZBYT) {
                mr16(0x00C0, opcod, &opd1);
                mchbvst((int)opd2.p2val);
            }
            else {
                mr16(0x00C1, opcod, &opd1);
                mchbvst((int)opd2.p2val);
                if(opd1.siz!=SIZWRD) errgen(45, NULL);
            }
        }
        else errgen(46, NULL);
    }
    else errgen(46, NULL);
}

static void op34c(void) {

    int         typ1,
                typ2;

    typ1=typudv(&opd1);
    typ2=chktyp(&opd2);
    if(typ1==TYPBRG) {
        if(chkcl(&opd2)) mr32(0x00D2, opcod, &opd1);
        else if(typ2==TYPNUM) {
            if(opd2.p2val == 1UL) mr32(0x00D0, opcod, &opd1);
            else {
                mr32(0x00C0, opcod, &opd1);
                mchbvst((int)opd2.p2val);
            }
        }
        else errgen(46, NULL);
    }
    else if(typ1==TYPWRG) {
        if(chkcl(&opd2)) mr32(0x00D3, opcod, &opd1);
        else if(typ2==TYPNUM) {
            if(opd2.p2val == 1UL) mr32(0x00D1, opcod, &opd1);
            else {
                mr32(0x00C1, opcod, &opd1);
                mchbvst((int)opd2.p2val);
            }
        }
        else errgen(46, NULL);
    }
    else if(typ1==TYPERG) {
        if(chkcl(&opd2)) {
            mchbvst(0x0066);
            mr32(0x00D3, opcod, &opd1);
        }
        else if(typ2==TYPNUM) {
            if(opd2.p2val == 1UL) {
                mchbvst(0x0066);
                mr32(0x00D1, opcod, &opd1);
            }
            else {
                mchbvst(0x0066);
                mr16(0x00C1, opcod, &opd1);
                mchbvst((int)opd2.p2val);
            }
        }
        else errgen(46, NULL);
    }
    else if(typ1==TYPVAR || typ1==TYPIND || typ1==TYPINX) {
        if(chkcl(&opd2)) {
            if(opd1.siz == SIZBYT) mr32(0x00D2, opcod, &opd1);
            else if(opd1.siz == SIZDWD) {
                mchbvst(0x0066);
                mr32(0x00D3, opcod, &opd1);
            }
            else {
                mr16(0x00D3, opcod, &opd1);
                if(opd1.siz!=SIZWRD) errgen(45, NULL);
            }
        }
        else if(typ2==TYPNUM && opd2.p2val==1UL) {
            if(opd1.siz == SIZBYT) mr16(0x00D0, opcod, &opd1);
            else if(opd1.siz == SIZDWD) {
                mchbvst(0x0066);
                mr32(0x00D1, opcod, &opd1);
            }
            else {
                mr16(0x00D1, opcod, &opd1);
                if(opd1.siz!=SIZWRD) errgen(45, NULL);
            }
        }
        else if(typ2==TYPNUM) {
            if(opd1.siz == SIZBYT) {
                mr16(0x00C0, opcod, &opd1);
                mchbvst((int)opd2.p2val);
            }
            else if(opd1.siz == SIZDWD) {
                mchbvst(0x0066);
                mr32(0x00C1, opcod, &opd1);
                mchbvst((int)opd2.p2val);
            }
            else {
                mr16(0x00C1, opcod, &opd1);
                mchbvst((int)opd2.p2val);
                if(opd1.siz!=SIZWRD) errgen(45, NULL);
            }
        }
        else errgen(46, NULL);
    }
    else errgen(46, NULL);
}

/* TEST */
static char *op35(char *pt) {

    int         typ1,
                typ2;

    pt=expr(pt);
    memcpy(&opd1, &expval, sizeof(struct opdv));
    pt=nxtod(pt);
    pt=expr(pt);
    memcpy(&opd2, &expval, sizeof(struct opdv));
    typ1=typudv(&opd1);
    typ2=chktyp(&opd2);
    if(prctyp<=2) {
        if(chkal(&opd1) && typ2==TYPNUM) {
            mchbvst(0x00A8);
            mchbvst((int)opd2.p2val);
        }
        else if(chkax(&opd1) && typ2==TYPNUM) {
            mchbvst(0x00A9);
            mchwvst((int)opd2.p2val);
        }
        else if(typ1==TYPBRG) {
            if(typ2==TYPNUM) {
                mr16(0x00F6, 0x0000, &opd1);
                mchbvst((int)opd2.p2val);
            }
            else if(typ2==TYPBRG)
                mr16(0x0084, (opd2.bsr & 0x00FF), &opd1);
            else errgen(46, NULL);
        }
        else if(typ1==TYPWRG) {
            if(typ2==TYPNUM) {
                mr16(0x00F7, 0x0000, &opd1);
                mchwvst((int)opd2.p2val);
            }
            else if(typ2==TYPWRG)
                mr16(0x0085, (opd2.bsr & 0x00FF), &opd1);
            else errgen(46, NULL);
        }
        else if(typ1==TYPVAR || typ1==TYPIND) {
            if(typ2==TYPBRG) {
                mr16(0x0084, (opd2.bsr & 0x00FF)<<3, &opd1);
                if(opd1.siz!=SIZBYT && opd1.siz!=SIZNON)
                    errgen(45, NULL);
            }
            else if(typ2==TYPWRG) {
                mr16(0x0085, (opd2.bsr & 0x00FF)<<3, &opd1);
                if(opd1.siz!=SIZWRD && opd1.siz!=SIZNON)
                    errgen(45, NULL);
            }
            else if(typ2==TYPNUM) {
                if(opd1.siz==SIZBYT) {
                    mr16(0x00F6, 0x0000, &opd1);
                    mchbvst((int)opd2.p2val);
                }
                else {
                    mr16(0x00F7, 0x0000, &opd1);
                    mchwvst((int)opd2.p2val);
                    if(opd1.siz!=SIZWRD) errgen(45, NULL);
                }
            }
            else errgen(46, NULL);
        }
        else errgen(46, NULL);
    }
    else {
        if(chkal(&opd1) && typ2==TYPNUM) {
            mchbvst(0x00A8);
            mchbvst((int)opd2.p2val);
        }
        else if(chkax(&opd1) && typ2==TYPNUM) {
            mchbvst(0x00A9);
            mchwvst((int)opd2.p2val);
        }
        else if(chkeax(&opd1) && typ2==TYPNUM) {
            mchbvst(0x0066);
            mchbvst(0x00A9);
            mchlvst(opd2.p2val);
        }
        else if(typ1==TYPBRG) {
            if(typ2==TYPNUM) {
                mr32(0x00F6, 0x0000, &opd1);
                mchbvst((int)opd2.p2val);
            }
            else if(typ2==TYPBRG)
                mr32(0x0084, (opd2.bsr & 0x00FF), &opd1);
            else errgen(46, NULL);
        }
        else if(typ1==TYPWRG) {
            if(typ2==TYPNUM) {
                mr32(0x00F7, 0x0000, &opd1);
                mchwvst((int)opd2.p2val);
            }
            else if(typ2==TYPWRG)
                mr32(0x0085, (opd2.bsr & 0x00FF), &opd1);
            else errgen(46, NULL);
        }
        else if(typ1==TYPERG) {
            if(typ2==TYPNUM) {
                mchbvst(0x0066);
                mr32(0x00F7, 0x0000, &opd1);
                mchlvst(opd2.p2val);
            }
```


アセンブラ用語集

ベースレジスタ　BXレジスタを指し，間接アドレス指定時にアドレスレジスタ，ベースレジスタとして使用することができる。

```
            else if(typ2==TYPERG) {
                mchbvst(0x0066);
                mr32(0x0085, (opd2.bsr & 0x00FF), &opd1);
            }
            else errgen(46, NULL);
        }
        else if(typ1==TYPVAR || typ1==TYPIND || typ1==TYPINX) {
            if(typ2==TYPBRG) {
                mr32(0x0084, (opd2.bsr & 0x00FF)<<3, &opd1);
                if(opd1.siz!=SIZBYT && opd1.siz!=SIZNON)
                    errgen(45, NULL);
            }
            else if(typ2==TYPWRG) {
                mr32(0x0085, (opd2.bsr & 0x00FF)<<3, &opd1);
                if(opd1.siz!=SIZWRD && opd1.siz!=SIZNON)
                    errgen(45, NULL);
            }
            else if(typ2==TYPERG) {
                mchbvst(0x0066);
                mr32(0x0085, (opd2.bsr & 0x00FF)<<3, &opd1);
                if(opd1.siz!=SIZDWD && opd1.siz!=SIZNON)
                    errgen(45, NULL);
            }
            else if(typ2==TYPNUM) {
                if(opd1.siz==SIZBYT) {
                    mr32(0x00F6, 0x0000, &opd1);
                    mchbvst((int)opd2.p2val);
                }
                else if(opd1.siz == SIZDWD) {
                    mchbvst(0x0066);
                    mr32(0x00F7, 0x0000, &opd1);
                    mchlvst(opd2.p2val);
                }
                else {
                    mr32(0x00F7, 0x0000, &opd1);
                    mchwvst((int)opd2.p2val);
                    if(opd1.siz!=SIZWRD) errgen(45, NULL);
                }
            }
            else errgen(46, NULL);
        }
        else errgen(46, NULL);
    }
    return pt;
}

/* XCHG */
static char *op36(char *pt) {

    int     typ1,
            typ2;

    pt=expr(pt);
    memcpy(&opd1, &expval, sizeof(struct opdv));
    pt=nxtod(pt);
    pt=expr(pt);
    memcpy(&opd2, &expval, sizeof(struct opdv));
    typ1=typudv(&opd1);
    typ2=typudv(&opd2);
    if(prctyp<=2) {
        if(chkax(&opd1) && typ2==TYPWRG)
            mchbvst(0x0090+(opd2.bsr & 0x00FF));
        else if(typ1==TYPWRG && chkax(&opd2))
            mchbvst(0x0090+(opd1.bsr & 0x00FF));
        else if(typ1==TYPBRG) {
            if(typ2==TYPBRG)
                mr16(0x0086, (opd1.bsr & 0x00FF)<<3, &opd2);
            else if(typ2==TYPVAR || typ2==TYPIND) {
                mr16(0x0086, (opd1.bsr & 0x00FF)<<3, &opd2);
                if(opd2.siz!=SIZBYT && opd2.siz!=SIZNON)
                    errgen(50, NULL);
            }
            else errgen(46, NULL);
        }
        else if(typ1==TYPWRG) {
            if(typ2==TYPWRG)
                mr16(0x0087, (opd1.bsr & 0x00FF)<<3, &opd2);
            else if(typ2==TYPVAR || typ2==TYPIND) {
                mr16(0x0087, (opd1.bsr & 0x00FF)<<3, &opd2);
                if(opd2.siz!=SIZWRD && opd2.siz!=SIZNON)
                    errgen(50, NULL);
            }
            else errgen(46, NULL);
        }
        else if(typ1==TYPVAR || typ1==TYPIND) {
            if(typ2==TYPBRG) {
                mr16(0x0086, (opd2.bsr & 0x00FF)<<3, &opd1);
                if(opd1.siz!=SIZBYT && opd1.siz!=SIZNON)
                    errgen(50, NULL);
            }
            else if(typ2==TYPWRG) {
                mr16(0x0087, (opd2.bsr & 0x00FF)<<3, &opd1);
                if(opd1.siz!=SIZWRD && opd1.siz!=SIZNON)
                    errgen(50, NULL);
            }
            else errgen(46, NULL);
        }
        else errgen(46, NULL);
    }
    else {
        if(chkax(&opd1) && typ2==TYPWRG)
            mchbvst(0x0090+(opd2.bsr & 0x00FF));
        else if(typ1==TYPWRG && chkax(&opd2))
            mchbvst(0x0090+(opd1.bsr & 0x00FF));
        else if(chkeax(&opd1) && typ2==TYPWRG) {
            mchbvst(0x0066);
            mchbvst(0x0090+(opd2.bsr & 0x00FF));
        }
        else if(typ1==TYPWRG && chkeax(&opd2)) {
            mchbvst(0x0066);
            mchbvst(0x0090+(opd1.bsr & 0x00FF));
        }
        else if(typ1==TYPBRG) {
            if(typ2==TYPBRG)
                mr32(0x0086, (opd1.bsr & 0x00FF)<<3, &opd2);
            else if(typ2==TYPVAR || typ2==TYPIND || typ2==TYPINX) {
                mr32(0x0086, (opd1.bsr & 0x00FF)<<3, &opd2);
                if(opd2.siz!=SIZBYT && opd2.siz!=SIZNON)
                    errgen(50, NULL);
            }
            else errgen(46, NULL);
        }
        else if(typ1==TYPWRG) {
            if(typ2==TYPWRG)
                mr32(0x0087, (opd1.bsr & 0x00FF)<<3, &opd2);
            else if(typ2==TYPVAR || typ2==TYPIND || typ2==TYPINX) {
                mr32(0x0087, (opd1.bsr & 0x00FF)<<3, &opd2);
                if(opd2.siz!=SIZWRD && opd2.siz!=SIZNON)
                    errgen(50, NULL);
            }
            else errgen(46, NULL);
        }
        else if(typ1==TYPERG) {
            if(typ2==TYPERG) {
                mchbvst(0x0066);
                mr32(0x0087, (opd1.bsr & 0x00FF)<<3, &opd2);
            }
            else if(typ2==TYPVAR || typ2==TYPIND || typ2==TYPINX) {
                mchbvst(0x0066);
                mr32(0x0087, (opd1.bsr & 0x00FF)<<3, &opd2);
                if(opd2.siz!=SIZDWD && opd2.siz!=SIZNON)
                    errgen(50, NULL);
            }
            else errgen(46, NULL);
        }
        else if(typ1==TYPVAR || typ1==TYPIND || typ1==TYPINX) {
            if(typ2==TYPBRG) {
                mr32(0x0086, (opd2.bsr & 0x00FF)<<3, &opd1);
                if(opd1.siz!=SIZBYT && opd1.siz!=SIZNON)
                    errgen(50, NULL);
            }
            else if(typ2==TYPWRG) {
                mr32(0x0087, (opd2.bsr & 0x00FF)<<3, &opd1);
                if(opd1.siz!=SIZWRD && opd1.siz!=SIZNON)
                    errgen(50, NULL);
            }
            else if(typ2==TYPERG) {
                mchbvst(0x0066);
                mr32(0x0087, (opd2.bsr & 0x00FF)<<3, &opd1);
                if(opd1.siz!=SIZDWD && opd1.siz!=SIZNON)
                    errgen(50, NULL);
            }
            else errgen(46, NULL);
        }
        else errgen(46, NULL);
    }
    return pt;
}

/* BOUND */
static char *op37(char *pt) {

    int     typ1,
            typ2;

    pt=expr(pt);
    memcpy(&opd1, &expval, sizeof(struct opdv));
    pt=nxtod(pt);
    pt=expr(pt);
    memcpy(&opd2, &expval, sizeof(struct opdv));
    typ1=chktyp(&opd1);
    typ2=chktyp(&opd2);
    if(passno==1 && typ2==TYPNUM && (opd2.att & DEF)==0) {
        typ2=TYPVAR;
        if(prctyp>=3) opd2.siz=SIZQWD;
        else opd2.siz=SIZDWD;
    }
    if(prctyp==1 || prctyp==2) {
        if(typ1==TYPWRG) {
            if(typ2==TYPVAR || typ2==TYPIND) {
                mr16(0x0062, (opd1.bsr & 0x00FF)<<3, &opd2);
                if(opd2.siz!=SIZDWD && opd2.siz!=SIZNON)
                    errgen(45, NULL);
            }
            else errgen(46, NULL);
        }
        else errgen(46, NULL);
    }
    else if(prctyp>=3) {
        if(typ1==TYPWRG) {
            if(typ2==TYPVAR || typ2==TYPIND || typ2==TYPINX) {
                mr32(0x0062, (opd1.bsr & 0x00FF)<<3, &opd2);
                if(opd2.siz!=SIZDWD && opd2.siz!=SIZNON)
                    errgen(45, NULL);
            }
            else errgen(46, NULL);
        }
        else if(typ1==TYPERG) {
            if(typ2==TYPVAR || typ2==TYPIND || typ2==TYPINX) {
                mchbvst(0x0066);
                mr32(0x0062, (opd1.bsr & 0x00FF)<<3, &opd2);
                if(opd2.siz!=SIZQWD && opd2.siz!=SIZNON)
                    errgen(45, NULL);
            }
            else errgen(46, NULL);
        }
        else errgen(46, NULL);
    }
    else errgen(41, NULL);
    return pt;
}

/* ENTER */
static char *op38(char *pt) {

    int     typ1,
            typ2;

    pt=expr(pt);
    memcpy(&opd1, &expval, sizeof(struct opdv));
    pt=nxtod(pt);
    pt=expr(pt);
    memcpy(&opd2, &expval, sizeof(struct opdv));
    typ1=chktyp(&opd1);
    typ2=chktyp(&opd2);
    if(prctyp>=1) {
        if(typ1==TYPNUM && typ2==TYPNUM) {
            mchbvst(0x00C8);
            mchwvst((int)opd1.p2val);
            mchbvst((int)opd2.p2val);
        }
        else errgen(46, NULL);
    }
    else errgen(41, NULL);
    return pt;
}
```

アセンブラ用語集

ポインタ　ストリングやメモリブロックなどで，直接データではなくデータを格納するアドレスを指す場合に使用する。

```c

/* INS */
static char *op39(char *pt) {

    pt=expr(pt);
    memcpy(&opd1, &expval, sizeof(struct opdv));
    pt=nxtod(pt);
    pt=expr(pt);
    memcpy(&opd2, &expval, sizeof(struct opdv));
    if(chkdx(&opd2)) {
        if(prctyp==1 || prctyp==2) {
            if(opd1.ovr != 0) errgen(49, NULL);
            if(opd1.siz==SIZBYT) mchbvst(0x006C);
            else {
                mchbvst(0x006D);
                if(opd1.siz!=SIZWRD) errgen(45, NULL);
            }
        }
        else if(prctyp>=3) {
            if(opd1.ovr != 0) errgen(49, NULL);
            if(opd1.siz==SIZBYT) mchbvst(0x006C);
            else if(opd1.siz == SIZDWD) {
                mchbvst(0x0066);
                mchbvst(0x006D);
            }
            else {
                mchbvst(0x006D);
                if(opd1.siz!=SIZWRD) errgen(45, NULL);
            }
        }
        errgen(41, NULL);
    }
    else errgen(46, NULL);
    return pt;
}

/* LAR, LSL */
static char *op40(char *pt) {

    int     typ1,
            typ2;

    pt=expr(pt);
    memcpy(&opd1, &expval, sizeof(struct opdv));
    pt=nxtod(pt);
    pt=expr(pt);
    memcpy(&opd2, &expval, sizeof(struct opdv));
    typ1=chktyp(&opd1);
    typ2=typudv(&opd2);
    if(prctyp==2) {
        if(typ1==TYPWRG) {
            if(typ2==TYPWRG)
                mr16(opcod, (opd1.bsr & 0x00FF)<<3, &opd2);
            else if(typ2==TYPVAR || typ2==TYPIND) {
                mr16(opcod, (opd1.bsr & 0x00FF)<<3, &opd2);
                if(opd2.siz!=SIZWRD && opd2.siz!=SIZNON)
                    errgen(50, NULL);
            }
            else errgen(46, NULL);
        }
        else errgen(46, NULL);
    }
    else if(prctyp>=3) {
        if(typ1==TYPWRG) {
            if(typ2==TYPWRG)
                mr32(opcod, (opd1.bsr & 0x00FF)<<3, &opd2);
            else if(typ2==TYPVAR || typ2==TYPIND || typ2==TYPINX) {
                mr32(opcod, (opd1.bsr & 0x00FF)<<3, &opd2);
                if(opd2.siz!=SIZWRD && opd2.siz!=SIZNON)
                    errgen(50, NULL);
            }
            else errgen(46, NULL);
        }
        else if(typ1==TYPERG) {
            if(typ2==TYPERG) {
                mchbvst(0x0066);
                mr32(opcod, (opd1.bsr & 0x00FF)<<3, &opd2);
            }
            else if(typ2==TYPVAR || typ2==TYPIND || typ2==TYPINX) {
                mchbvst(0x0066);
                mr32(opcod, (opd1.bsr & 0x00FF)<<3, &opd2);
                if(opd2.siz!=SIZDWD && opd2.siz!=SIZNON)
                    errgen(50, NULL);
            }
            else errgen(46, NULL);
        }
    }
    return pt;
}

/* OUTS */
static char *op41(char *pt) {

    pt=expr(pt);
    memcpy(&opd1, &expval, sizeof(struct opdv));
    pt=nxtod(pt);
    pt=expr(pt);
    memcpy(&opd2, &expval, sizeof(struct opdv));
    opd2.ovr=segcnv(opd2.ovr);
    if(chkdx(&opd1)) {
        if(prctyp==1 || prctyp==2) {
            if(opd2.ovr) mchbvst(opd2.ovr);
            if(opd2.siz == SIZBYT) mchbvst(0x006E);
            else {
                mchbvst(0x006F);
                if(opd2.siz!=SIZWRD) errgen(45, NULL);
            }
        }
        else if(prctyp>=3) {
            if(opd2.ovr) mchbvst(opd2.ovr);
            if(opd2.siz == SIZBYT) mchbvst(0x006E);
            else if(opd2.siz == SIZDWD) {
                mchbvst(0x0066);
                mchbvst(0x006F);
            }
            else {
                mchbvst(0x006F);
                if(opd2.siz!=SIZWRD) errgen(45, NULL);
            }
        }
        else errgen(41, NULL);
    }
    else errgen(46, NULL);
    return pt;
}

/* ARPL */
static char *op42(char *pt) {

    int     typ1,
            typ2;

    pt=expr(pt);
    memcpy(&opd1, &expval, sizeof(struct opdv));
    pt=nxtod(pt);
    pt=expr(pt);
    memcpy(&opd2, &expval, sizeof(struct opdv));
    typ1=typudv(&opd1);
    typ2=chktyp(&opd2);
    if(prctyp==2) {
        if(typ2==TYPWRG) {
            if(typ1==TYPWRG)
                mr16(0x0063, (opd2.bsr & 0x00FF)<<3, &opd1);
            else if(typ1==TYPVAR || typ1==TYPIND) {
                mr16(0x0063, (opd2.bsr & 0x00FF)<<3, &opd1);
                if(opd1.siz!=SIZWRD && opd1.siz!=SIZNON)
                    errgen(50, NULL);
            }
            else errgen(46, NULL);
        }
        else errgen(46, NULL);
    }
    else if(prctyp>=3) {
        if(typ2==TYPWRG) {
            if(typ1==TYPWRG)
                mr32(0x0063, (opd2.bsr & 0x00FF)<<3, &opd1);
            else if(typ1==TYPVAR || typ1==TYPIND || typ1==TYPINX) {
                mr32(0x0063, (opd2.bsr & 0x00FF)<<3, &opd1);
                if(opd1.siz!=SIZWRD && opd1.siz!=SIZNON)
                    errgen(50, NULL);
            }
            else errgen(46, NULL);
        }
        else errgen(46, NULL);
    }
    else errgen(42, NULL);
    return pt;
}

/* BSF, BSR */
static char *op43(char *pt) {

    int     typ1,
            typ2;

    pt=expr(pt);
    memcpy(&opd1, &expval, sizeof(struct opdv));
    pt=nxtod(pt);
    pt=expr(pt);
    memcpy(&opd2, &expval, sizeof(struct opdv));
    typ1=chktyp(&opd1);
    typ2=typudv(&opd2);
    if(prctyp>=3) {
        if(typ1==TYPWRG) {
            if(typ2==TYPWRG) mr32(opcod, (opd1.bsr & 0x00FF)<<3, &opd2);
            else if(typ2==TYPVAR || typ2==TYPIND) {
                mr32(opcod, (opd1.bsr & 0x00FF)<<3, &opd2);
                if(opd2.siz!=SIZWRD && opd2.siz!=SIZNON)
                    errgen(50, NULL);
            }
            else errgen(46, NULL);
        }
        else if(typ1==TYPERG) {
            if(typ2==TYPERG) {
                mchbvst(0x0066);
                mr32(opcod, (opd1.bsr & 0x00FF)<<3, &opd2);
            }
            else if(typ2==TYPVAR || typ2==TYPIND || typ2==TYPINX) {
                mchbvst(0x0066);
                mr32(opcod, (opd1.bsr & 0x00FF)<<3, &opd2);
                if(opd2.siz!=SIZDWD && opd2.siz!=SIZNON)
                    errgen(50, NULL);
            }
            else errgen(46, NULL);
        }
        else errgen(46, NULL);
    }
    else errgen(43, NULL);
    return pt;
}

/* BT, BTC, BTR, BTS */
static char *op44(char *pt) {

    int     typ1,
            typ2,
            reg;

    pt=expr(pt);
    memcpy(&opd1, &expval, sizeof(struct opdv));
    pt=nxtod(pt);
    pt=expr(pt);
    memcpy(&opd2, &expval, sizeof(struct opdv));
    typ1=typudv(&opd1);
    typ2=chktyp(&opd2);
    if(prctyp>=3) {
        reg=opcod & 0x00FF;
        opcod=(opcod >> 8) & 0x00FF;
        if(typ2==TYPNUM) {
            opcod=0x0FBA;
            if(typ1==TYPWRG) {
                mr32(opcod, reg, &opd1);
                mchbvst((int)opd2.p2val);
            }
            else if(typ1==TYPERG) {
                mchbvst(0x0066);
                mr32(opcod, reg, &opd1);
                mchbvst((int)opd2.p2val);
            }
            else if(typ1==TYPVAR || typ1==TYPIND || typ1==TYPINX) {
                if(opd1.siz == SIZDWD) {
                    mchbvst(0x0066);
                    mr32(opcod, reg, &opd1);
                    mchbvst((int)opd2.p2val);
                }
                else {
```


アセンブラ用語集
マクロ　プログラム中に繰り返し使われる命令ブロックをマクロ定義し，アセンブルするとき対応する命令ブロックに置き換える．

```
                    mr32(opcod, reg, &opd1);
                    mchbvst((int)opd2.p2val);
                    if(opd1.siz!=SIZWRD) errgen(45, NULL);
                }
            }
            else errgen(46, NULL);
        }
        else if(typ2==TYPWRG) {
            opcod|=0x0F00;
            if(typ1==TYPWRG)
                mr32(opcod, (opd2.bsr & 0x00FF)<<3, &opd1);
            else if(typ1==TYPVAR || typ1==TYPIND) {
                mr32(opcod, (opd2.bsr & 0x00FF)<<3, &opd1);
                if(opd1.siz!=SIZWRD && opd1.siz!=SIZNON)
                    errgen(45, NULL);
            }
            else errgen(46, NULL);
        }
        else if(typ2==TYPERG) {
            opcod|=0x0F00;
            if(typ1==TYPERG) {
                mchbvst(0x0066);
                mr32(opcod, (opd2.bsr & 0x00FF)<<3, &opd1);
            }
            else if(typ1==TYPVAR || typ1==TYPIND || typ1==TYPINX) {
                mchbvst(0x0066);
                mr32(opcod, (opd2.bsr & 0x00FF)<<3, &opd1);
                if(opd1.siz!=SIZDWD && opd1.siz!=SIZNON)
                    errgen(45, NULL);
            }
            else errgen(46, NULL);
        }
        else errgen(46, NULL);
    }
    else errgen(43, NULL);
    return pt;
}

/* LFS, LGS, LSS */
static char *op45(char *pt) {

    int     typ1,
            typ2;

    pt=expr(pt);
    memcpy(&opd1, &expval, sizeof(struct opdv));
    pt=nxtod(pt);
    pt=expr(pt);
    memcpy(&opd2, &expval, sizeof(struct opdv));
    typ1=chktyp(&opd1);
    typ2=typudf(&opd2);
    if(prctyp>=3) {
        if(typ1==TYPWRG) {
            if(typ2==TYPVAR || typ2==TYPIND || typ2==TYPINX) {
                mr32(opcod, (opd1.bsr & 0x00FF)<<3, &opd2);
                if(opd2.siz!=SIZDWD && opd2.siz!=SIZNON)
                    errgen(45, NULL);
            }
            else errgen(46, NULL);
        }
        else if(typ1==TYPERG) {
            if(typ2==TYPVAR || typ2==TYPIND || typ2==TYPINX) {
                mchbvst(0x0066);
                mr32(opcod, (opd1.bsr & 0x00FF)<<3, &opd2);
                if(opd2.siz!=SIZFWD && opd2.siz!=SIZNON)
                    errgen(45, NULL);
            }
            else errgen(46, NULL);
        }
        else errgen(46, NULL);
    }
    else errgen(43, NULL);
    return pt;
}

/* MOVSX, MOVZX */
static char *op46(char *pt) {

    int     typ1,
            typ2;

    pt=expr(pt);
    memcpy(&opd1, &expval, sizeof(struct opdv));
    pt=nxtod(pt);
    pt=expr(pt);
    memcpy(&opd2, &expval, sizeof(struct opdv));
    typ1=chktyp(&opd1);
    typ2=typudv(&opd2);
    if(prctyp>=3) {
        if(typ1==TYPWRG) {
            if(typ2==TYPBRG) mr32(opcod, (opd1.bsr & 0x00FF)<<3, &opd2);
            else if(typ2==TYPVAR || typ2==TYPIND || typ2==TYPINX) {
                mr32(opcod, (opd1.bsr & 0x00FF)<<3, &opd2);
                if(opd2.siz!=SIZBYT && opd2.siz!=SIZNON)
                    errgen(45, NULL);
            }
            else errgen(46, NULL);
        }
        else if(typ1==TYPERG) {
            if(typ2==TYPWRG) {
                mchbvst(0x0066);
                mr32(opcod+1, (opd1.bsr & 0x00FF)<<3, &opd2);
            }
            else if(typ2==TYPVAR || typ2==TYPIND || typ2==TYPINX) {
                mchbvst(0x0066);
                mr32(opcod+1, (opd1.bsr & 0x00FF)<<3, &opd2);
                if(opd2.siz!=SIZWRD && opd2.siz!=SIZNON)
                    errgen(45, NULL);
            }
            else errgen(46, NULL);
        }
        else errgen(46, NULL);
    }
    return pt;
}

/* CMPXCHG, XADD */
static char *op47(char *pt) {

    int     typ1,
            typ2;

    pt=expr(pt);
    memcpy(&opd1, &expval, sizeof(struct opdv));
    pt=nxtod(pt);
    pt=expr(pt);
    memcpy(&opd2, &expval, sizeof(struct opdv));
    typ1=typudv(&opd1);
    typ2=chktyp(&opd2);
    if(prctyp>=4) {
        if(typ2==TYPBRG) {
            if(typ1==TYPBRG) mr32(opcod, (opd2.bsr & 0x00FF)<<3, &opd1);
            else if(typ1==TYPVAR || typ1==TYPIND || typ1==TYPINX) {
                mr32(opcod, (opd2.bsr & 0x00FF)<<3, &opd1);
                if(opd1.siz!=SIZBYT && opd1.siz!=SIZNON)
                    errgen(50, NULL);
            }
            else errgen(46, NULL);
        }
        else if(typ2==TYPWRG) {
            if(typ1==TYPWRG) mr32(opcod, (opd2.bsr & 0x00FF)<<3, &opd1);
            else if(typ1==TYPVAR || typ1==TYPIND || typ1==TYPINX) {
                mr32(opcod+1, (opd2.bsr & 0x00FF)<<3, &opd1);
                if(opd1.siz!=SIZWRD && opd1.siz!=SIZNON)
                    errgen(50, NULL);
            }
            else errgen(46, NULL);
        }
        else if(typ2==TYPERG) {
            if(typ1==TYPERG) {
                mchbvst(0x0066);
                mr32(opcod, (opd2.bsr & 0x00FF)<<3, &opd1);
            }
            else if(typ1==TYPVAR || typ1==TYPIND || typ1==TYPINX) {
                mchbvst(0x0066);
                mr32(opcod+1, (opd2.bsr & 0x00FF)<<3, &opd1);
                if(opd1.siz!=SIZDWD && opd1.siz!=SIZNON)
                    errgen(50, NULL);
            }
            else errgen(46, NULL);
        }
        else errgen(46, NULL);
    }
    else errgen(44, NULL);
    return pt;
}

/* IMUL */
static char *op48(char *pt) {

    pt=expr(pt);
    memcpy(&opd1, &expval, sizeof(struct opdv));
    if(isend(pt)) op48a();
    else {
        pt=nxtod(pt);
        pt=expr(pt);
        memcpy(&opd2, &expval, sizeof(struct opdv));
        if(isend(pt)) op48b();
        else {
            pt=nxtod(pt);
            pt=expr(pt);
            memcpy(&opd3, &expval, sizeof(struct opdv));
            op48c();
        }
    }
    return pt;
}

static void op48a(void) {

    int     typ;

    typ=typudv(&opd1);
    if(prctyp<=2) {
        if(typ==TYPBRG) mr16(0x00F6, 0x0028, &opd1);
        else if(typ==TYPWRG) mr16(0x00F7, 0x0028, &opd1);
        else if(typ==TYPVAR || typ==TYPIND) {
            if(opd1.siz == SIZBYT) mr16(0x00F6, 0x0028, &opd1);
            else {
                mr16(0x00F7, 0x0028, &opd1);
                if(opd1.siz!=SIZWRD && opd1.siz!=SIZNON)
                    errgen(45, NULL);
            }
        }
        else errgen(46, NULL);
    }
    else {
        if(typ==TYPBRG) mr32(0x00F6, 0x0028, &opd1);
        else if(typ==TYPWRG) mr32(0x00F7, 0x0028, &opd1);
        else if(typ==TYPERG) {
            mchbvst(0x0066);
            mr32(0x00F7, 0x0028, &opd1);
        }
        else if(typ==TYPVAR || typ==TYPIND || typ==TYPINX) {
            if(opd1.siz == SIZBYT) mr32(0x00F6, 0x0028, &opd1);
            else if(opd1.siz == SIZDWD) {
                mchbvst(0x0066);
                mr32(0x00F7, 0x0028, &opd1);
            }
            else {
                mr32(0x00F7, 0x0028, &opd1);
                if(opd1.siz!=SIZWRD) errgen(45, NULL);
            }
        }
        else errgen(46, NULL);
    }
}

static void op48b(void) {

    int     typ1,
            typ2;

    if(prctyp>=3) {
        typ1=chktyp(&opd1);
        typ2=typudv(&opd2);
        if(typ1==TYPWRG) {
            if(typ2==TYPWRG)
                mr32(0x0FAF, (opd1.bsr & 0x00FF)<<3, &opd2);
            else if(typ2==TYPVAR || typ2==TYPIND || typ2==TYPINX) {
                mr32(0x0FAF, (opd1.bsr & 0x00FF)<<3, &opd2);
                if(opd2.siz!=SIZWRD && opd2.siz!=SIZNON)
                    errgen(50, NULL);
            }
            else errgen(46, NULL);
        }
```

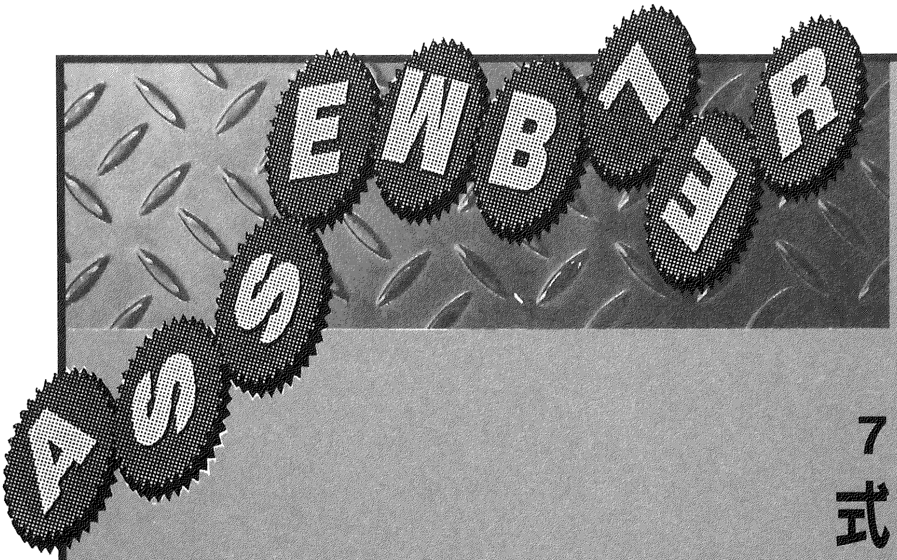
```
            else if(typ1==TYPERG) {
                if(typ2==TYPERG) {
                    mchbvst(0x0066);
                    mr32(0x0FAF, (opd1.bsr & 0x00FF)<<3, &opd2);
                }
                else if(typ2==TYPVAR || typ2==TYPIND || typ2==TYPINX) {
                    mchbvst(0x0066);
                    mr32(0x0FAF, (opd1.bsr & 0x00FF)<<3, &opd2);
                    if(opd2.siz!=SIZDWD && opd2.siz!=SIZNON)
                        errgen(50, NULL);
                }
                else errgen(46, NULL);
            }
            else errgen(46, NULL);
        }
        else errgen(43, NULL);
}

static void op48c(void) {

        int     typ1,
                typ2,
                typ3;

        typ1=chktyp(&opd1);
        typ2=typudv(&opd2);
        typ3=chktyp(&opd3);
        if(typ3==TYPNUM) {
            if(prctyp==1 || prctyp==2) {
                if(typ1==TYPWRG) {
                    if(typ2==TYPWRG) {
                        if(chkis8(&opd3)) {
                            mr16(0x006B, (opd1.bsr & 0x00FF)<<3, &opd2);
                            mchbvst((int)opd3.p2val);
                            return;
                        }
                        else {
                            mr16(0x0069, (opd1.bsr & 0x00FF)<<3, &opd2);
                            mchwvst((int)opd3.p2val);
                            return;
                        }
                    }
                    else if(typ2==TYPVAR || typ2==TYPIND) {
                        if(chkis8(&opd3)) {
                            mr16(0x006B, (opd1.bsr & 0x00FF)<<3, &opd2);
                            mchbvst((int)opd3.p2val);
                        }
                        else {
                            mr16(0x0069, (opd1.bsr & 0x00FF)<<3, &opd2);
                            mchwvst((int)opd3.p2val);
                        }
                        if(opd2.siz!=SIZWRD && opd2.siz!=SIZNON)
                            errgen(50, NULL);
                        return;
                    }
                    else errgen(46, NULL);
                }
                else errgen(46, NULL);
            }
            else if(prctyp>=3) {
                if(typ1==TYPWRG) {
                    if(typ2==TYPWRG) {
                        if(chkis8(&opd3)) {
                            mr32(0x006B, (opd1.bsr & 0x00FF)<<3, &opd2);
                            mchbvst((int)opd3.p2val);
                            return;
                        }
                        else {
                            mr32(0x0069, (opd1.bsr & 0x00FF)<<3, &opd2);
                            mchwvst((int)opd3.p2val);
                            return;
                        }
                    }
                    else if(typ2==TYPVAR || typ2==TYPIND || typ2==TYPINX) {
                        if(chkis8(&opd3)) {
                            mr32(0x006B, (opd1.bsr & 0x00FF)<<3, &opd2);
                            mchbvst((int)opd3.p2val);
                        }
                        else {
                            mr32(0x0069, (opd1.bsr & 0x00FF)<<3, &opd2);
                            mchwvst((int)opd3.p2val);
                        }
                        if(opd2.siz!=SIZWRD && opd2.siz!=SIZNON)
                            errgen(50, NULL);
                        return;
                    }
                    else errgen(46, NULL);
                }
                if(typ1==TYPERG) {
                    if(typ2==TYPERG) {
                        mchbvst(0x0066);
                        if(chkis8(&opd3)) {
                            mr32(0x006B, (opd1.bsr & 0x00FF)<<3, &opd2);
                            mchbvst((int)opd3.p2val);
                            return;
                        }
                        else {
                            mr32(0x0069, (opd1.bsr & 0x00FF)<<3, &opd2);
                            mchlvst(opd3.p2val);
                            return;
                        }
                    }
                    else if(typ2==TYPVAR || typ2==TYPIND || typ2==TYPINX) {
                        mchbvst(0x0066);
                        if(chkis8(&opd3)) {
                            mr32(0x006B, (opd1.bsr & 0x00FF)<<3, &opd2);
                            mchbvst((int)opd3.p2val);
                        }
                        else {
                            mr32(0x0069, (opd1.bsr & 0x00FF)<<3, &opd2);
                            mchwvst((int)opd3.p2val);
                        }
                        if(opd2.siz!=SIZDWD && opd2.siz!=SIZNON)
                            errgen(50, NULL);
                        return;
                    }
                    else errgen(46, NULL);
                }
                else errgen(46, NULL);
            }
            else errgen(41, NULL);
        }
        else errgen(46, NULL);
}

/* SHLD, SHRD */
static char *op49(char *pt) {

        int     typ1,
                typ2,
                typ3,
                reg;

        pt=expr(pt);
        memcpy(&opd1, &expval, sizeof(struct opdv));
        pt=nxtod(pt);
        pt=expr(pt);
        memcpy(&opd2, &expval, sizeof(struct opdv));
        pt=nxtod(pt);
        pt=expr(pt);
        memcpy(&opd3, &expval, sizeof(struct opdv));
        typ1=typudv(&opd1);
        typ2=chktyp(&opd2);
        typ3=chktyp(&opd3);
        reg=(opd2.bsr & 0x00FF)<<3;
        if(prctyp>=3) {
            if(typ3==TYPNUM) {
                if(typ2==TYPWRG) {
                    if(typ1==TYPWRG) {
                        mr32(opcod, reg, &opd1);
                        mchbvst((int)opd3.p2val);
                    }
                    else if(typ1==TYPVAR || typ1==TYPIND || typ1==TYPINX) {
                        mr32(opcod, reg, &opd1);
                        mchbvst((int)opd3.p2val);
                        if(opd1.siz!=SIZWRD && opd1.siz!=SIZNON)
                            errgen(50, NULL);
                    }
                    else errgen(46, NULL);
                }
                else if(typ2==TYPERG) {
                    if(typ1==TYPERG) {
                        mchbvst(0x0066);
                        mr32(opcod, reg, &opd1);
                        mchbvst((int)opd3.p2val);
                    }
                    else if(typ1==TYPVAR || typ1==TYPIND || typ1==TYPINX) {
                        mchbvst(0x0066);
                        mr32(opcod, reg, &opd1);
                        mchbvst((int)opd3.p2val);
                        if(opd1.siz!=SIZDWD && opd1.siz!=SIZNON)
                            errgen(50, NULL);
                    }
                    else errgen(46, NULL);
                }
                else errgen(46, NULL);
            }
            else if(chkcl(&opd3)) {
                if(typ2==TYPWRG) {
                    if(typ1==TYPWRG) mr32(opcod+1, reg, &opd1);
                    else if(typ1==TYPVAR || typ1==TYPIND || typ1==TYPINX) {
                        mr32(opcod+1, reg, &opd1);
                        if(opd1.siz!=SIZWRD && opd1.siz!=SIZNON)
                            errgen(50, NULL);
                    }
                    else errgen(46, NULL);
                }
                else if(typ2==TYPERG) {
                    if(typ1==TYPERG) {
                        mchbvst(0x0066);
                        mr32(opcod+1, reg, &opd1);
                    }
                    else if(typ1==TYPVAR || typ1==TYPIND || typ1==TYPINX) {
                        mchbvst(0x0066);
                        mr32(opcod+1, reg, &opd1);
                        if(opd1.siz!=SIZDWD && opd1.siz!=SIZNON)
                            errgen(50, NULL);
                    }
                    else errgen(46, NULL);
                }
                else errgen(46, NULL);
            }
            else errgen(46, NULL);
        }
        else errgen(43, NULL);
        return pt;
}

/* .186 */
static char *op50(char *pt) {

        if(prefix) errgen(52, NULL);
        locbuf[0]=(char)'¥0';
        prctyp=1;
        if(strlen(lblbuf)) errgen(51, NULL);
        return pt;
}

/* .286 */
static char *op51(char *pt) {

        if(prefix) errgen(52, NULL);
        locbuf[0]=(char)'¥0';
        prctyp=2;
        if(strlen(lblbuf)) errgen(51, NULL);
        return pt;
}

/* .386 */
static char *op52(char *pt) {

        if(prefix) errgen(52, NULL);
        locbuf[0]=(char)'¥0';
        prctyp=3;
        if(strlen(lblbuf)) errgen(51, NULL);
        return pt;
}

/* .486 */
static char *op53(char *pt) {

        if(prefix) errgen(52, NULL);
        locbuf[0]=(char)'¥0';
        prctyp=4;
        if(strlen(lblbuf)) errgen(51, NULL);
        return pt;
```

アセンブラ用語集
メモリモデル　インテル社の80x86CPUなどのようにセグメントという概念から派生したメモリの管理方法

```
}

/* .8086 */
static char *op54(char *pt) {

    if(prefix) errgen(52, NULL);
    locbuf[0]=(char)'¥0';
    prctyp=0;
    if(strlen(lblbuf)) errgen(51, NULL);
    return pt;
}

/* DB */
static char *op55(char *pt) {

    int     typ,
            chr;
    char    *wk;

    if(prefix) errgen(52, NULL);
    while(1) {
        wk=token(pt);
        chr=(int)tknbuf[0];
        if(chr=='¥'') {
            pt=wk;
            wk=&tknbuf[1];
            while(1) {
                chr=(int)*wk;
                if(chr=='¥0') {
                    errgen(32, NULL);
                    break;
                }
                if(chr=='¥'') {
                    wk++;
                    chr=(int)*wk;
                    if(chr!='¥'') break;
                    mchbvst(chr);
                }
                else mchbvst(chr);
                wk++;
            }
        }
        else {
            pt=expr(pt);
            typ=chktyp(&expval);
            if(typ==TYPNUM) mchbvst((int)expval.p2val);
            else {
                errgen(46, NULL);
                break;
            }
        }
        if(isend(pt)) break;
        pt=nxtod(pt);
    }
    lblreg(lac, VAR, SIZBYT);
    return pt;
}

/* DW */
static char *op56(char *pt) {

    int     typ;

    if(prefix) errgen(52, NULL);
    while(1) {
        pt=expr(pt);
        typ=chktyp(&expval);
        if(typ!=TYPNUM && typ!=TYPVAR && typ!=TYPLBL) {
            errgen(46, NULL);
            break;
        }
        mchwvst((int)expval.p2val);
        if(isend(pt)) break;
        pt=nxtod(pt);
    }
    lblreg(lac, VAR, SIZWRD);
    return pt;
}

/* DD */
static char *op57(char *pt) {

    int     typ;

    if(prefix) errgen(52, NULL);
    while(1) {
        pt=expr(pt);
        typ=chktyp(&expval);
        if(typ!=TYPNUM && typ!=TYPVAR && typ!=TYPLBL) {
            errgen(46, NULL);
            break;
        }
        mchlvst(expval.p2val);
        if(isend(pt)) break;
        pt=nxtod(pt);
    }
    lblreg(lac, VAR, SIZDWD);
    return pt;
}

/* ELSE */
static char *op58(char *pt) {

    if(prefix) errgen(52, NULL);
    locbuf[0]=(char)'¥0';
    if(ifcnt) {
        if(ifflg[ifcnt-1]) ifflg[ifcnt-1]=FALSE;
        else ifflg[ifcnt-1]=TRUE;
    }
    if(strlen(lblbuf)) errgen(51, NULL);
    return pt;
}

/* END */
static char *op59(char *pt) {

    if(prefix) errgen(52, NULL);
    eopgm=TRUE;
    if(strlen(lblbuf)) errgen(51, NULL);
    return pt;
}

/* ENDIF */
static char *op60(char *pt) {

    if(prefix) errgen(52, NULL);
    locbuf[0]=(char)'¥0';
    if(ifcnt) ifcnt--;
    if(strlen(lblbuf)) errgen(51, NULL);
    return pt;
}

/* EQU */
static char *op61(char *pt) {

    int             typ,
                    chr,
                    len;
    unsigned long lx;

    if(prefix) errgen(52, NULL);
    locbuf[0]=(char)'¥0';
    pt=expr(pt);
    typ=chktyp(&expval);
    if(passno==1) lx=expval.p1val;
    else lx=expval.p2val;
    if(expval.att & DEF) {
        if(typ==TYPNUM || typ==TYPVAR || typ==TYPLBL) {
            chr=(int)lblbuf[0];
            if(chr=='¥0') errgen(53, NULL);
            else {
                len=strlen(lblbuf);
                chr=lblbuf[len-1];
                if(chr==':') errgen(54, NULL);
                else {
                    lblreg(lx, expval.att, expval.siz);
                    sprintf(mchbuf, "= %08lX", lx);
                }
            }
        }
        else errgen(46, NULL);
    }
    return pt;
}

/* IF */
static char *op62(char *pt) {

    int             typ;
    unsigned long lx;

    if(prefix) errgen(52, NULL);
    pt=expr(pt);
    typ=chktyp(&expval);
    if(passno==1) lx=expval.p1val;
    else lx=expval.p2val;
    if(expval.att & DEF) {
        if(typ==TYPNUM) {
            if(lx!=EXPFLS) ifflg[ifcnt]=TRUE;
            else ifflg[ifcnt]=FALSE;
            ifcnt++;
        }
        else ftlgen(2);
    }
    else ftlgen(1);
    if(strlen(lblbuf)) errgen(51, NULL);
    return pt;
}

/* ORG */
static char *op63(char *pt) {

    int             typ;
    unsigned long lx;

    if(prefix) errgen(52, NULL);
    pt=expr(pt);
    typ=chktyp(&expval);
    if(passno==1) lx=expval.p1val;
    else lx=expval.p2val;
    lx&=0x0000FFFFUL;
    if(expval.att & DEF) {
        if(typ==TYPNUM || typ==TYPVAR || typ==TYPLBL) {
            objorg(lx);
            sprintf(mchbuf, "+ %08lX", lx-lac);
            lac=lx;
        }
        else ftlgen(2);
    }
    else ftlgen(1);
    if(strlen(lblbuf)) errgen(51, NULL);
    return pt;
}
```

アセンブラ用語集

ラベル　プログラムの中で数値や文字列，そしてアドレスを参照したときに使われる名前のこと。



7 式

特集 アセンブラの作り方

簡易アセンブラではオペランド＝式として，オペランドの解析をすべてexpr関数に任せている．したがって，式を評価するということは，オペランドに何が書かれているかを解析して演算が必要であれば演算を行い，レジスタ修飾のような特殊な要素が書かれていればそれをコード化することである．プログラム7（EXP86.C）は，このような処理を行う関数群である．

7-1 レジスタテーブル

レジスタ記号は，レジスタ直接とレジスタ修飾アドレッシングモードを表すオペランドに使用され，式の一部である．レジスタテーブルの1つの要素は，レジスタ記号の文字列とレジスタ識別コードから構成される．regtbl構造体が，このレジスタテーブルの構造を表す．レジスタ識別コードは，上位8ビットがレジスタの種別を表す内部コードで，下位8ビットが表5のレジスタコードになっている．なお，レジスタテーブルはシンボルテーブルや命令テーブルと異なりハッシングされていない．このテーブルは，式の要素を調べる際，isreg関数によって使用される．

7-2 演算子の優先順位とオペランドの属性

簡易アセンブラで使用できる演算子と，その演算子のオペランドに書くことのできる要素の属性および演算結果の属性を表11にまとめてある．表11の演算子の優先順位は，MASMのものと同様である．もちろん，簡易アセンブラで使用できない演算子はこの表の中にはない．MASMと多少違っているのは，絶対形式のアセンブラであるため，NEAR PTRやFAR PTRの演算にナンバーが使用できることと，：の演算にナンバーによるセグメントの指定ができることである．

7-3 式の処理方法

このアセンブラの式の評価方法は，マイクロコンピュータの黎明期にパロ・アルト版のタイニーBASICで使用された方法と基本的に同じものである．あまり能率がよいとはいえないが，わかりやすいのが特徴である．ただ，BASICのインタープリタでは数値だけ処理すればよいが，表11にもあるように属性やレジスタ修飾におけるレジスタの処理，およびセグメントオーバーライドなどの処理も行わなければならない．

expr関数は式の評価の入り口で，この関数を呼び出すと，引数のポインタの示す位置からオペランドの解析を実

アセンブラ用語集
リロケータブル　メモリ上のどの位置にロードしても実行可能なプログラムをいい，アドレスインディペンドも同じ．

行し，結果をexpval構造体に返す．この構造体の内容は，expr関数に都合のよい構造をしているので，オペランドがどのタイプであるかを知るためには適していない．オペランドのタイプを知るためには，chktyp関数を使用する．また，この結果をmdregmemバイトとscixrbsrバイトに変換して機械コードを出力するには，mr16やmr32の関数を利用する．

expr関数は，それぞれの優先順位を持つ演算を実行する関数に分かれている．これらの関数は，優先順位の低い関数から高い関数へと順次呼び出されていき，最後に要素の値を求める関数を呼び出す．簡易アセンブラで実質的に値を持つ要素は，2進数，10進数，16進数，文字列，LACの現在値，レジスタ記号およびシンボルの参照である．

要素の値が求められると，今度は逆に呼び出された逆順をたどって処理しなければならない順位の演算子を処理する関数まで戻って演算が行われ，その結果がさらに低い優先順位を持つ演算子を処理する関数に渡される．そして処理すべき演算子が発見できなくなるとexpr関数へ戻る．

以上の処理を疑似的にC言語風にまとめると，各レベルの演算処理の関数は次のように書くことができる（図7）．

表11　演算子の優先順位とオペランドの属性および演算結果の属性

優先順位	演　算　子	第1オペランド	第2オペランド	結　果
0	:	CS/DS/ES/SS/FS/GS	VAR/LBL	VAR/LBL
		CS/DS/ES/SS/FS/GS	NUM	VAR
		CS/DS/ES/SS/FS/GS	IND	IND
		NUM	VAR	FAR
1	OFFSET	VAR/LBL/NUM/FAR		NUM
	NEAR PTR	VAR/LBL/NUM/FAR		LBL
	FAR PTR	FAR		FAR
	BYTE PTR	VAR/LBL/NUM		VAR/NUM
	BYTE PTR	IND		IND
	xWORD PTR	VAR/LBL/NUM		VAR/NUM
	xWORD PTR	IND		IND
2	+	VAR/LBL/NUM/FAR		VAR/LBL/NUM/FAR
	−	NUM		NUM
3	*	NUM	NUM	NUM
		EREG	NUM	SCALED INDEX
	/,MOD,SHL,	NUM	NUM	NUM
	SHR	NUM	NUM	NUM
4	+	NUM(IND)	NUM(IND)	NUM(IND)
		NUM(IND)	VAR/LBL(IND)	VAR/LBL(IND)
		VAR/LBL(IND)	NUM	VAR/LBL(IND)
	−	NUM(IND)	NUM	NUM
		VAR/LBL(IND)	NUM	VAR/LBL(IND)
		VAR/LBL(IND)	VAR/LBL(IND)	NUM(IND)
5	GT,GE,EQ,	NUM	NUM	NUM
	NE,LE,LT	VAR/LBL	VAR/LBL	NUM
6	NOT	NUM	NUM	
7	AND	NUM	NUM	NUM
8	OR,XOR	NUM	NUM	NUM

注：NUM＝ナンバー，VAR＝バリアブル，LBL＝ラベル，IND＝レジスタ修飾，FAR＝ポインタ

図7

```
2項演算子の処理() {
        1つ優先順位の高い演算子の処理を呼び出す;
        expvalをローカル変数に保存する;
        while(1) {
            if(次の演算子が自分のレベルの演算子でない) break;
            演算子をローカル変数に保存;
            1つ優先順位の高い演算子の処理を呼び出す;
            保存されているexpvalと直前に返されたexpvalとの
            間で保存した演算子に従って演算を行い結果をローカル
            変数に保存する;
        }
        ローカル変数に保存されている値をexpvalに戻す;
}

単項演算子の処理() {
        if(次の演算子が自分のレベルの演算子でない)
                1つ優先順位の高い演算子の処理を呼び出す;
        else {
                1つ優先順位の高い演算子の処理を呼び出す;
                expvalに対して演算を行う;
        }
}
```

リスト7 EXP86.C

```c
/******************************************/
/*  式の評価(AS86)                         */
/******************************************/

#include <STDLIB.H>
#include <STRING.H>
#include <CTYPE.H>
#include "AS86.H"

#define VALNUM 0          /* NUMBER */
#define VALNER 1          /* NEAR LABEL OR VARIABLE */
#define VALFAR 2          /* FAR LABEL */
#define VALREG 3          /* REGISTER */
#define VALIND 4          /* INDIRECT */

struct regtbl {           /* REGISTER TABLEの形式 */
    char    name[4];         /* REGISTERの名前 */
    int     code;         /* REGISTERの識別CODE */
};

/* REGISTER TABLE */
struct regtbl reglst[44] = {
     {"AL",  BREG+0}, {"CL",  BREG+1}, {"DL",  BREG+2}, {"BL",  BREG+3},
     {"AH",  BREG+4}, {"CH",  BREG+5}, {"DH",  BREG+6}, {"BH",  BREG+7},
     {"AX",  WREG+0}, {"CX",  WREG+1}, {"DX",  WREG+2}, {"BX",  WREG+3},
     {"SP",  WREG+4}, {"BP",  WREG+5}, {"SI",  WREG+6}, {"DI",  WREG+7},
     {"ES",  SREG+0}, {"CS",  SREG+1}, {"SS",  SREG+2}, {"DS",  SREG+3},
     {"FS",  SREG+4}, {"GS",  SREG+5}, {"EAX", EREG+0}, {"ECX", EREG+1},
     {"EDX", EREG+2}, {"EBX", EREG+3}, {"ESP", EREG+4}, {"EBP", EREG+5},
     {"ESI", EREG+6}, {"EDI", EREG+7}, {"CR0", CREG+0}, {"CR2", CREG+2},
     {"CR3", CREG+3}, {"DR0", DREG+0}, {"DR1", DREG+1}, {"DR2", DREG+2},
     {"DR3", DREG+3}, {"DR6", DREG+6}, {"DR7", DREG+7}, {"TR3", TREG+3},
     {"TR4", TREG+4}, {"TR5", TREG+5}, {"TR6", TREG+6}, {"TR7", TREG+7}
};

static char * _near expror(char *pt);
static char * _near exprand(char *pt);
static char * _near exprnot(char *pt);
static char * _near exprcmp(char *pt);
static char * _near expras(char *pt);
static char * _near exprmd(char *pt);
static char * _near exprsgn(char *pt);
static char * _near exprptr(char *pt);
static char * _near exprsgm(char *pt);
static char * _near exprfct(char *pt);
static int _near isnumb(char *pt);
static int _near islac(char *pt);
static int _near isstrng(char *pt);
static int _near chkval(struct opdv *xx);
static int _near chkdsp(struct opdv *xx);
static int _near chkixr(struct opdv *xx);
static void _near opdadd(struct opdv *xx, struct opdv *yy);
static void _near opdsub(struct opdv *xx, struct opdv *yy);
static void _near chkcmb(struct opdv *xx);

/* 式の評価の入り口 */
char *expr(char *pt) {

    char        *wk;
    int         chr;
    struct opdv   xx;

    experr=NORMAL;
    brackt=0;
    pt=expror(pt);
    memcpy(&xx, &expval, sizeof(struct opdv));
    while(1) {
        wk=token(pt);
        if(strcmp(tknbuf, "[")) break;
        pt=expror(pt);
        if(experr!=NORMAL) break;
        opdadd(&xx, &expval);
    }
    chr=(int)tknbuf[0];
    if(chr!='¥0' && chr!=',' && chr!=';') {
        errgen(35, NULL);
        pt=wk;
        while(1) {
            wk=token(pt);
            chr=(int)tknbuf[0];
            if(chr=='¥0' || chr==',' || chr==';') break;
            pt=wk;
        }
    }
    memcpy(&expval, &xx, sizeof(struct opdv));
    chkcmb(&expval);
    return pt;
}

/* OR式 */
static char * _near expror(char *pt) {

    char        *wk,
                op[4];
    struct opdv   xx;

    pt=exprand(pt);
    if(experr!=NORMAL) return pt;
    memcpy(&xx, &expval, sizeof(struct opdv));
    while(1) {
        wk=token(pt);
        if(stricmp(tknbuf, "OR") && stricmp(tknbuf, "XOR")) break;
        strcpy(op, tknbuf);
        pt=exprand(wk);
        if(experr!=NORMAL) break;
        if((expval.att & DEF)==0) xx.att &= ~DEF;
        if(chkval(&xx)!=VALNUM || chkval(&expval)!=VALNUM)
            errgen(8, NULL);
        if(stricmp(op, "OR")==0) {
            /* OR */
            xx.p1val |= expval.p1val;
            xx.p2val |= expval.p2val;
        }
        else {
            /* XOR */
            xx.p1val ^= expval.p1val;
            xx.p2val ^= expval.p2val;
        }
        xx.att = (xx.att & DEF)|NUM;
        xx.siz = 0;
        xx.bsr = 0;
        xx.ixr = 0;
        xx.ovr = 0;
        xx.scl = 0;
    }
    memcpy(&expval, &xx, sizeof(struct opdv));
    return pt;
}

/* AND式 */
static char * _near exprand(char *pt) {

    char        *wk;
    struct opdv   xx;

    pt=exprnot(pt);
    if(experr!=NORMAL) return pt;
    memcpy(&xx, &expval, sizeof(struct opdv));
    while(1) {
        wk=token(pt);
        if(stricmp(tknbuf, "AND")) break;
        pt=exprnot(wk);
        if((expval.att&DEF)==0) xx.att&=~DEF;
        if(experr!=NORMAL) break;
        if(chkval(&xx)!=VALNUM || chkval(&expval)!=VALNUM)
            errgen(8, NULL);
        /* AND */
        xx.p1val &= expval.p1val;
        xx.p2val &= expval.p2val;
        xx.att=(xx.att & DEF)|NUM;
        xx.siz = 0;
        xx.bsr = 0;
        xx.ixr = 0;
        xx.ovr = 0;
        xx.scl = 0;
    }
    memcpy(&expval, &xx, sizeof(struct opdv));
    return pt;
}

/* NOT式 */
static char * _near exprnot(char *pt) {

    char        *wk;

    wk=token(pt);
    if(stricmp(tknbuf, "NOT")==0) {
        pt=exprcmp(wk);
        if(chkval(&expval)!=VALNUM) errgen(9, NULL);
        /* NOT */
        expval.p1val = ~expval.p1val;
        expval.p2val = ~expval.p2val;
        expval.att = (expval.att & DEF)|NUM;
        expval.siz = 0;
        expval.bsr = 0;
        expval.ixr = 0;
        expval.ovr = 0;
        expval.scl = 0;
        return pt;
    }
    else return exprcmp(pt);
}

/* 比較式 */
static char * _near exprcmp(char *pt) {

    char        *wk,
                op[4];
    struct opdv   xx;

    pt=expras(pt);
    if(experr!=NORMAL) return pt;
    memcpy(&xx, &expval, sizeof(struct opdv));
    while(1) {
        wk=token(pt);
        if(stricmp(tknbuf, "GT") && stricmp(tknbuf, "GE") &&
            stricmp(tknbuf, "EQ") && stricmp(tknbuf, "NE") &&
            stricmp(tknbuf, "LE") && stricmp(tknbuf, "LT"))
            break;
        strcpy(op, tknbuf);
        pt=expras(wk);
        if(experr!=NORMAL) break;
        if((expval.att & DEF)==0) xx.att &= ~DEF;
        if(chkval(&xx) != chkval(&expval)) errgen(10, NULL);
        else if(chkval(&xx)!=VALNUM && chkval(&xx)!=VALNER)
            errgen(10, NULL);
        if(stricmp(op, "GT")==0) {
            /* GT */
            if(xx.p1val > expval.p1val) xx.p1val = EXPTRU;
            else xx.p1val = EXPFLS;
            if(xx.p2val > expval.p2val) xx.p2val = EXPTRU;
            else xx.p2val = EXPFLS;
```


アセンブラ用語集
ローカルラベル　同一のプロシージャ内でのみ有効なラベル．大規模なプログラムなどを作成するときラベル名の重なる心配がない．

```c
        }
        else if(stricmp(op, "GE")==0) {
            /* GE */
            if(xx.p1val >= expval.p1val) xx.p1val = EXPTRU;
            else xx.p1val = EXPFLS;
            if(xx.p2val >= expval.p2val) xx.p2val = EXPTRU;
            else xx.p2val = EXPFLS;
        }
        else if(stricmp(op, "EQ")==0) {
            /* EQ */
            if(xx.p1val == expval.p1val) xx.p1val = EXPTRU;
            else xx.p1val = EXPFLS;
            if(xx.p2val == expval.p2val) xx.p2val = EXPTRU;
            else xx.p2val = EXPFLS;
        }
        else if(stricmp(op, "NE")==0) {
            /* NE */
            if(xx.p1val != expval.p1val) xx.p1val = EXPTRU;
            else xx.p1val = EXPFLS;
            if(xx.p2val != expval.p2val) xx.p2val = EXPTRU;
            else xx.p2val = EXPFLS;
        }
        else if(stricmp(op, "LE")==0) {
            /* LE */
            if(xx.p1val <= expval.p1val) xx.p1val = EXPTRU;
            else xx.p1val = EXPFLS;
            if(xx.p2val <= expval.p2val) xx.p2val = EXPTRU;
            else xx.p2val = EXPFLS;
        }
        else {
            /* LT */
            if(xx.p1val > expval.p1val) xx.p1val = EXPTRU;
            else xx.p1val = EXPFLS;
            if(xx.p2val > expval.p2val) xx.p2val = EXPTRU;
            else xx.p2val = EXPFLS;
        }
        xx.att = (xx.att & DEF)|NUM;
        xx.siz = 0;
        xx.bsr = 0;
        xx.ixr = 0;
        xx.ovr = 0;
        xx.scl = 0;
    }
    memcpy(&expval, &xx, sizeof(struct opdv));
    return pt;
}

/* 加減式 */
static char * _near expras(char *pt) {

    char        *wk;
    int         op;
    struct opdv   xx;

    pt=exprmd(pt);
    if(experr!=NORMAL) return pt;
    memcpy(&xx, &expval, sizeof(struct opdv));
    while(1) {
        wk=token(pt);
        if(strcmp(tknbuf, "+") && strcmp(tknbuf, "-")) break;
        op=(int)tknbuf[0];
        pt=exprmd(wk);
        if(experr!=NORMAL) break;
        /* X+Y */
        if(op == '+') opdadd(&xx, &expval);
        /* X-Y */
        else opdsub(&xx, &expval);
    }
    memcpy(&expval, &xx, sizeof(struct opdv));
    return pt;
}

/* 乗除式 */
static char * _near exprmd(char *pt) {

    char            *wk,
                    op[4];
    unsigned long lx;
    struct opdv     xx;

    pt=exprsgn(pt);
    if(experr!=NORMAL) return pt;
    memcpy(&xx, &expval, sizeof(struct opdv));
    while(1) {
        wk=token(pt);
        if(strcmp(tknbuf, "*") && strcmp(tknbuf, "/") &&
            stricmp(tknbuf, "MOD") && stricmp(tknbuf, "SHR")
            && stricmp(tknbuf, "SHL")) break;
        strcpy(op, tknbuf);
        pt=exprsgn(wk);
        if(strcmp(op, "*")==0) {
            /* X*Y */
            if(chkixr(&xx)==NORMAL && chkval(&expval)==VALNUM) {
                /* SCALE */
                if(passno!=3) lx = expval.p1val;
                else lx = expval.p2val;
                if(lx!=1L && lx!=2L && lx!=4L && lx!=8L) {
                    errgen(16, NULL);
                    xx.scl = 1;
                }
                else xx.scl = (unsigned int)lx;
                if((xx.bsr & 0xFF00)!=EREG || (xx.bsr & 0x00FF)==4)
                    errgen(34, NULL);
                else {
                    xx.ixr = xx.bsr;
                    xx.bsr = 0;
                }
            }
            else if(chkval(&xx)==VALNUM && chkval(&expval)==VALNUM) {
                /* NUMBER*NUMBER */
                xx.p1val *= expval.p1val;
                xx.p2val *= expval.p2val;
                xx.att=(xx.att & DEF)|NUM;
                xx.siz = 0;
                xx.bsr = 0;
                xx.ixr = 0;
                xx.ovr = 0;
            }
            else {
                /* OPERAND ERROR */
                errgen(17, NULL);
                xx.p1val *= expval.p1val;
                xx.p2val *= expval.p2val;
                xx.att = (xx.att & DEF) | NUM;
                xx.siz = 0;
                xx.bsr = 0;
                xx.ixr = 0;
                xx.ovr = 0;
            }
        }
        else {
            if(chkval(&xx)!=VALNUM ||
                chkval(&expval)!=VALNUM) errgen(18, NULL);
            if(strcmp(op, "/")==0) {
                /* X/Yの場合 */
                if(expval.p1val != 0) xx.p1val /= expval.p1val;
                if(expval.p2val == 0) errgen(19, NULL);
                else xx.p2val /= expval.p2val;
            }
            else if(stricmp(op, "MOD")==0) {
                /* X MOD Yの場合 */
                if(expval.p1val != 0) xx.p1val %= expval.p2val;
                if(expval.p2val==0) errgen(19, NULL);
                else xx.p2val %= expval.p2val;
            }
            else if(stricmp(op, "SHR")==0) {
                /* X SHR Yの場合 */
                xx.p1val >>= expval.p1val;
                xx.p2val >>= expval.p2val;
            }
            else {
                /* X SHL Yの場合 */
                xx.p1val <<= expval.p1val;
                xx.p2val <<= expval.p2val;
            }
            xx.att=(xx.att & DEF)|NUM;
            xx.siz = 0;
            xx.bsr = 0;
            xx.ixr = 0;
            xx.ovr = 0;
        }
    }
    memcpy(&expval, &xx, sizeof(struct opdv));
    return pt;
}

/* 符号式 */
static char * _near exprsgn(char *pt) {

    char    *wk;
    int     op;

    wk=token(pt);
    if(strcmp(tknbuf, "+")==0 || strcmp(tknbuf, "-")==0) {
        op=(int)tknbuf[0];
        pt=exprptr(wk);
        if(op=='-') {
            /* -Xの場合 */
            if(chkval(&expval)!=VALNUM) errgen(20, NULL);
            expval.p1val = -expval.p1val;
            expval.p2val = -expval.p2val;
            expval.att = (expval.att & DEF) | NUM;
            expval.siz = 0;
            expval.bsr = 0;
            expval.ixr = 0;
            expval.ovr = 0;
        }
        /* +Xの場合は何もしない */
        return pt;
    }
    return exprptr(pt);
}

/* PTR式 */
static char * _near exprptr(char *pt) {

    char        *wk,
                op[8];

    wk=token(pt);
    if(stricmp(tknbuf, "OFFSET") && stricmp(tknbuf, "BYTE") &&
        stricmp(tknbuf, "WORD") && stricmp(tknbuf, "DWORD") &&
        stricmp(tknbuf, "FWORD") && stricmp(tknbuf, "QWORD") &&
        stricmp(tknbuf, "NEAR") && stricmp(tknbuf, "FAR"))
            return exprsgm(pt);
    strcpy(op, tknbuf);
    /* OFFSET以外の場合はPTRがあることを確認 */
    if(stricmp(op, "OFFSET")) {
        wk=token(wk);
        if(stricmp(tknbuf, "PTR")) return exprsgm(pt);
    }
    pt=exprsgm(wk);
    /* OFFSETの場合 */
    if(stricmp(op, "OFFSET")==0) {
        if(chkval(&expval)==VALREG) errgen(21, NULL);
        expval.att = (expval.att & DEF)|NUM;
        expval.siz = 0;
        expval.bsr = 0;
        expval.ixr = 0;
        expval.ovr = 0;
```

```
    }
    /* NEAR PTRの場合 */
    else if(stricmp(op, "NEAR")==0) {
        if(chkval(&expval)!=VALNUM && chkval(&expval)!=VALNER) {
            expval.bsr = 0;
            expval.ixr = 0;
            errgen(22, NULL);
        }
        expval.att = (expval.att & DEF)|LBL;
        expval.siz = SIZNER;
    }
    /* FAR PTRの場合 */
    else if(stricmp(op, "FAR")==0) {
        if(chkval(&expval)!=VALFAR) errgen(23, NULL);
        expval.att = (expval.att & DEF)|LBL;
        expval.siz = SIZFAR;
    }
    /* BYTEまたはxWORD PTRの場合 */
    else {
        if(chkval(&expval)==VALREG || chkval(&expval)==VALFAR) {
            expval.bsr = 0;
            expval.ixr = 0;
            errgen(24, NULL);
        }
        expval.att = (expval.att & DEF)|VAR;
        if(stricmp(op, "BYTE")==0) expval.siz = SIZBYT;
        else if(stricmp(op, "WORD")==0) expval.siz = SIZWRD;
        else if(stricmp(op, "DWORD")==0) expval.siz = SIZDWD;
        else if(stricmp(op, "FWORD")==0) expval.siz = SIZFWD;
        else expval.siz = SIZQWD;
    }
    return pt;
}

/* SEGMENT式 */
static char * _near exprsgm(char *pt) {

    char            *wk;
    unsigned int    rx;
    struct opdv     xx;

    pt=exprfct(pt);
    if(experr!=NORMAL) return pt;
    memcpy(&xx, &expval, sizeof(struct opdv));
    while(1) {
        wk=token(pt);
        if(strcmp(tknbuf, ":")) break;
        pt=exprfct(wk);
        if(experr!=NORMAL) break;
        if((expval.att&DEF)==0) xx.att&=~DEF;
        /* 絶対値PAR式 */
        if(chkval(&xx)==VALNUM && chkval(&expval)==VALNUM) {
            if(passno!=3) xx.aseg=(unsigned int)xx.p1val;
            else xx.aseg = (unsigned int)xx.p2val;
            xx.p1val = expval.p1val;
            xx.p2val = expval.p2val;
            xx.siz = SIZFAR;
        }
        /* SEGMENT OVERRIDE */
        else if(chkval(&xx)==VALREG && (xx.bsr&0xFF00)==SREG) {
            rx = xx.bsr;
            memcpy(&xx, &expval, sizeof(struct opdv));
            if(xx.ovr) errgen(13, NULL);
            else xx.ovr = rx;
            if((xx.att & SPC)==NUM) xx.att = (xx.att & DEF)|VAR;
        }
        /* 上記以外のX:Y */
        else {
            errgen(25, NULL);
            memset(&xx, 0, sizeof(struct opdv));
            xx.att = (xx.att & DEF)|NUM;
        }
    }
    memcpy(&expval, &xx, sizeof(struct opdv));
    return pt;
}

/* 要素 */
static char * _near exprfct(char *pt) {

    char            *wk;
    int             chr;

    wk=token(pt);
    chr=(int)tknbuf[0];
    if(strchr("*+,-/:;)]", chr)!=NULL) {
        /* 不正な要素 */
        errgen(26, NULL);
        experr=ERROR;
        memset(&expval, 0, sizeof(struct opdv));
        expval.att=NUM;
        return pt;
    }
    pt=wk;
    if(strcmp(tknbuf,"(")==0) {
        /* (式) */
        pt=expror(pt);
        if(experr==NORMAL) {
            wk=token(pt);
            if(strcmp(tknbuf, ")")) {
                experr=ERROR;
                errgen(27, NULL);
            }
            else pt=wk;
        }
        return pt;
    }
    if(strcmp(tknbuf, "[")==0) {
        /* [式] */
        if(brackt) errgen(28, NULL);
        brackt++;
        pt=expror(pt);
        if(experr==NORMAL) {
            wk=token(pt);
            if(strcmp(tknbuf, "]")) {
                experr=ERROR;
                errgen(29, NULL);
            }
            else {
                pt=wk;
                brackt--;
            }
        }
        return pt;
    }
    /* 数値 */
    if(isnumb(tknbuf)) return pt;
    /* LACの現在値 */
    if(islac(tknbuf)) return pt;
    /* 文字列 */
    if(isstrng(tknbuf)) return pt;
    /* REGISTER */
    if(isreg(tknbuf)) return pt;
    /* 以上の場合以外はSYMBOLとみなす */
    lblev(tknbuf);
    return pt;
}

/* 数値 */
static int _near isnumb(char *pt) {

    char            *wk;
    int             chr,
                    len;
    unsigned long ly;

    chr=(int)*pt;
    if(isascii(chr)==FALSE || isdigit(chr)==FALSE) return FALSE;
    len=strlen(pt);
    chr=(int)*(pt+len-1);
    if(chr=='b' || chr=='B') {      /* 2進数 */
        ly=strtoul(pt, &wk, 2);
        if(stricmp(wk, "B")) errgen(30, NULL);
    }
    else if(chr=='h' || chr=='H') {   /* 16進数 */
        ly=strtoul(pt, &wk, 16);
        if(stricmp(wk, "H")) errgen(31, NULL);
    }
    else {                          /* 10進数 */
        ly=strtoul(pt, &wk, 10);
        chr=(int)*wk;
        if(stricmp(wk, "D") && chr!='¥0') errgen(32, NULL);
    }
    memset(&expval, 0, sizeof(struct opdv));
    expval.p1val = ly;
    expval.p2val = ly;
    expval.att = DEF|NUM;
    return TRUE;
}

/* LAC */
static int _near islac(char *pt) {

    if(strcmp(pt, "$")) return FALSE;
    memset(&expval, 0, sizeof(struct opdv));
    expval.att = DEF|LBL;
    expval.p1val = lac;
    expval.p2val = lac;
    return TRUE;
}

/* 文字列 */
static int _near isstrng(char *pt) {

    char    *wk;
    int     chr,
            cnt;
    union {
        char            cx[4];
        unsigned long lx;
    } xx;

    chr=(int)*pt;
    if(chr!='¥'') return FALSE;
    wk=pt+1;
    cnt=0;
    xx.lx=0L;
    while(1) {
        chr=(int)*wk++;
        if(chr=='¥0') {
            experr=ERROR;
            errgen(33, pt);
            break;
        }
        if(chr=='¥'') {
            chr=(int)*wk;
            if(chr!='¥'') break;
            wk++;
        }
        if(cnt<4) {
            xx.cx[cnt] = (char)chr;
            cnt++;
        }
    }
    memset(&expval, 0, sizeof(struct opdv));
    expval.p1val = xx.lx;
    expval.p2val = xx.lx;
    expval.att = DEF|NUM;
    return TRUE;
}
```


アセンブラ用語集

ワード　16ビット（2バイト）で表せる値をワードといい，全角コードを表すのに使われる．

```c

/* REGISTER */
int isreg(char *pt) {

    int             cd,
                    ii;
    struct regtbl *rp;

    for(ii=0, rp=reglst; ii<44; ii++, rp++) {
        if(stricmp(pt, rp->name)==0) {
            memset(&expval, 0, sizeof(struct opdv));
            if(brackt) {
                expval.att = DEF|NUM;
                cd = rp->code;
                if((cd & 0xFF00)==WREG) {
                    cd&=0x00FF;
                    if(cd==3 || cd==5) expval.bsr = rp->code;
                    else if(cd==6 || cd==7) expval.ixr = rp->code;
                    else errgen(33, pt);
                }
                else if((cd & 0xFF00)==EREG) expval.bsr = cd;
                else errgen(33, pt);
            }
            else {
                expval.bsr = rp->code;
                expval.att = DEF|REG;
            }
            return TRUE;
        }
    }
    return FALSE;
}

/* 値の属性の調査 */
static int _near chkval(struct opdv *xx) {

    int   xat;

    xat=xx->att & SPC;
    if(xat==REG) return VALREG;
    if(xx->bsr || xx->ixr) return VALIND;
    if(xx->siz == SIZFAR) return VALFAR;
    if(xat == NUM) return VALNUM;
    return VALNER;
}

/* 変位のみの属性の調査 */
static int _near chkdsp(struct opdv *xx) {

    if((xx->att & SPC)==NUM) return VALNUM;
    return VALNER;
}

static int _near chkixr(struct opdv *xx) {

    if((xx->att & SPC)!=NUM) return ERROR;
    if(xx->p1val!=0 || xx->p2val!=0) return ERROR;
    if(xx->ixr!=0 || xx->scl!=0 || xx->ovr!=0) return ERROR;
    if(xx->bsr==0) return ERROR;
    return NORMAL;
}

/* 式の値の加算 */
static void _near opdadd(struct opdv *xx, struct opdv *yy) {

    int   xat,
        yat;

    xat=chkval(xx);
    yat=chkval(yy);
    if(xat==VALIND) xat=chkdsp(xx);
    if(yat==VALIND) yat=chkdsp(yy);
    xx->p1val += yy->p1val;
    xx->p2val += yy->p2val;
    if((yy->att & DEF)==0) xx->att &= ~DEF;
    if(yy->bsr) {
        if(xx->bsr == 0) xx->bsr = yy->bsr;
        else if((yy->bsr & 0xFF00)==EREG && (yy->bsr & 0x00FF)!=4 &&
            xx->ixr == 0) xx->ixr = yy->bsr;
        else errgen(11, NULL);
    }
    if(yy->ixr) {
        if(xx->ixr == 0) xx->ixr = yy->ixr;
        else errgen(12, NULL);
    }
    if(yy->ovr) {
        if(xx->ovr == 0) xx->ovr = yy->ovr;
        else errgen(13, NULL);
    }
    if(yy->scl) {
        if(xx->scl == 0) xx->scl = yy->scl;
        else errgen(12, NULL);
    }
    if(xat==VALNUM && yat==VALNER) {
        xx->att = yy->att;
        xx->siz = yy->siz;
    }
    if(xat==VALFAR || yat==VALFAR || xat==VALREG || yat==VALREG) {
        errgen(14, NULL);
        xx->att = (xx->att & DEF)|NUM;
        xx->siz = 0;
        xx->bsr = 0;
        xx->ixr = 0;
        xx->ovr = 0;
        xx->scl = 0;
    }
    else if(xat==VALNER && yat==VALNER) errgen(14, NULL);
}

/* 式の値の減算 */
static void _near opdsub(struct opdv *xx, struct opdv *yy) {

    int   xat,
        yat;

    xat=chkval(xx);
    if(xat==VALIND) xat=chkdsp(xx);
    yat=chkval(yy);
    if(yat==VALIND) errgen(15, NULL);
    xx->p1val -= yy->p1val;
    xx->p2val -= yy->p2val;
    if(xat==VALNER && yat==VALNER) {
        xx->att = (xx->att & DEF)|NUM;
        xx->siz = 0;
    }
    if((yy->att & DEF)==0) xx->att &= ~DEF;
    if(xat==VALFAR || xat==VALREG || yat==VALFAR || yat==VALREG)
        errgen(15, NULL);
    if(xat==VALNUM && yat==VALNER) errgen(15, NULL);
}

/* REGISTER修飾の正当性の調査と不要なOVERRIDEの削除 */
static void _near chkcmb(struct opdv *xx) {

    int     xat,
            bat,
            bvl,
            iat,
            ivl,
            wrk;

    xat = xx->att & SPC;
    if(xat==REG) return;
    if(xx->bsr) {
        bat = xx->bsr & 0xFF00;
        bvl = xx->bsr & 0x00FF;
        if(bat==WREG) {
            if(xx->ixr) {
                iat = xx->ixr & 0xFF00;
                ivl = xx->ixr & 0x00FF;
                if(iat!=WREG) {
                    errgen(38, NULL);
                    xx->ixr = 0;
                }
                else if(bvl==3 || bvl==5) {     /* BX or BP */
                    if(ivl!=6 && ivl!=7) {      /* SI or DI */
                        errgen(38, NULL);
                        xx->ixr = 0;
                    }
                }
                else if(bvl==6 || bvl==7) {     /* SI or DI */
                    if(ivl!=3 && ivl!=5) {      /* BX or BP */
                        errgen(38, NULL);
                        xx->ixr = 0;
                    }
                    else {
                        wrk = xx->bsr;
                        xx->bsr = xx->ixr;
                        xx->ixr = wrk;
                        wrk = bvl;
                        bvl = ivl;
                        ivl = wrk;
                    }
                }
            }
            if(bvl==5 && xx->ovr == SREG+2) xx->ovr = 0;    /* SS:[BP] */
            else if(xx->ovr == SREG+3) xx->ovr = 0;         /* DS: */
        }
        else {
            if(xx->ixr) {
                iat = xx->ixr & 0xFF00;
                ivl = xx->ixr & 0x00FF;
                if(iat!=EREG) {
                    errgen(38, NULL);
                    xx->ixr = 0;
                }
                else if(ivl==4) {           /* [xx+ESP*n] */
                    errgen(38, NULL);
                    xx->ixr = 0;
                }
            }
            if((bvl==4 || bvl==5) && xx->ovr==SREG+2) xx->ovr=0;
                                                /* SS:[ESP], SS:[EBP] */
            else if(xx->ovr == SREG+3) xx->ovr = 0;         /* DS: */
        }
        return;
    }
    if(xx->ixr) {
        iat = xx->ixr & 0xFF00;
        ivl = xx->ixr & 0x00FF;
        if(iat == WREG) {
            xx->bsr = xx->ixr;
            xx->ixr = 0;
            if(ivl==5 && xx->ovr == SREG+2) xx->ovr = 0;
            else if(xx->ovr == SREG+3) xx->ovr = 0;
        }
        else {
            if(xx->scl) errgen(39, NULL);
            xx->bsr = xx->ixr;
            xx->ixr = 0;
            xx->scl = 0;
            if((ivl==4 || ivl==5) && xx->ovr==SREG+2) xx->ovr = 0;
                                                /* SS:[ESP], SS:[EBP */
            else if(xx->ovr == SREG+3) xx->ovr = 0;         /* DS: */
        }
        return;
    }
    if(xx->ovr == SREG+3) xx->ovr = 0;                      /* DS: */
}

```

アセンブラ用語集

割り込みフラグ　ハードウェアの割り込みを制御するフラグ．1で割り込みを受け付け，0で受け付けない．

A まとめ

プログラムとこの原稿を含めて約1か月の間に製作したアセンブラなので，多々不備はあることと思うが，アセンブラのマクロ機能やリロケーションの処理などを除いて，アセンブラを製作するための重要なことは説明したつもりである．不備な点は，読者自身のトレーニングと思ってソースを自由に改造していただきたい．また，ほかのCPUに移植してみるのも面白いと思う．

本文では説明できなかったが，付録のフロッピーディスクには，TEST.ASMという入力した文字列を逆順に並べて表示する簡易アセンブラ用のサンプルプログラムが入っている．アセンブルして実行してみると，このアセンブラの特性が感じられると思う．なお，このアセンブラはバージョン2.1以降のMS-DOSが走行し，アセンブラのための動作領域が250KB程度あれば機種を選ばず実行できるはずである．

説明の最後に，まとめとして簡易アセンブラの文法をBNF記法で示しておく．この記法は完全なものではなく，禁止される組み合わせなどを十分に表現していない．また，レジスタ，命令，あるいは演算子は大文字と小文字の区別はないが，それも表現していない．さらに，演算子文字列と定義されている演算子には前後に空白文字が必要である．これらの点を注意して利用していただきたい．

表12　BNF記法による簡易アセンブラの文法

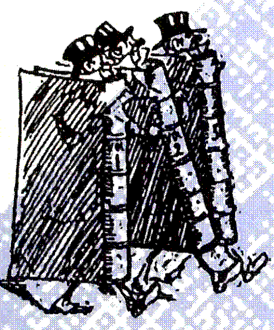
```
空白文字      := { SP | TAB }
終端記号      := LF
一般記号      := { ! | # | % | & | . | < | = | > | ? | @ |
                  ¥ | ^ | _ | ` | { | | | } | ￣ | ｡ | ｢ |
                  ｣ | ､ | ･ | - | ﾞ | ﾟ }
ドル記号      := $
引用記号      := '
引用文字      := ''
記号演算子    := { * | + | - | / }
区切記号      := { " | ( | ) | , | : | ; | [ | ] }
数字          := { 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 }
一般文字      := { A | B | C | D | E | F | G | H | I | J |
                  K | L | M | N | O | P | Q | R | S | T |
                  U | V | W | X | Y | Z | a | b | c | d |
                  e | f | g | h | i | j | k | l | m | n |
                  o | p | q | r | s | t | u | v | w | x |
                  y | z | ｦ | ｧ | ｨ | ｩ | ｪ | ｫ | ｬ | ｭ |
                  ｮ | ｯ | ｱ | ｲ | ｳ | ｴ | ｵ | ｶ | ｷ | ｸ |
                  ｹ | ｺ | ｻ | ｼ | ｽ | ｾ | ｿ | ﾀ | ﾁ | ﾂ |
                  ﾃ | ﾄ | ﾅ | ﾆ | ﾇ | ﾈ | ﾉ | ﾊ | ﾋ | ﾌ |
                  ﾍ | ﾎ | ﾏ | ﾐ | ﾑ | ﾒ | ﾓ | ﾔ | ﾕ | ﾖ |
                  ﾗ | ﾘ | ﾙ | ﾚ | ﾛ | ﾜ | ﾝ }
シンボル文字  := { ドル記号 | 一般記号 | 一般文字 | 数字 }
文字列用文字  := { 空白文字 | 一般記号 | 演算記号 | 区切記号 |
                  引用文字 | 数字 | 一般文字 }
記号・文字    := { 空白文字 | 一般記号 | ドル記号 | 引用記号 |
                  演算記号 | 区切記号 | 数字 | 一般文字 }

シンボル文字列:= シンボル文字 【 シンボル文字 】
プロセッサ命令:= 表1～表4のニモニックのいずれか
ディレクティブ:= { .186 | .286 | .386 | .486 | .8086 | DB |
                  DD | DW | ELSE | END | ENDIF | EQU | IF |
                  ORG }
属性演算子    := { OFFSET | { NEAR | FAR | BYTE | WORD | DWORD |
                  FWORD | QWORD } PTR }
演算子文字列  := { 属性演算子 | AND | EQ | GE | GT | LE | LT |
                  MOD | NE | OR | SHL | SHR | XOR }
2進数列       := { 0 | 1 } 【 2進数列 】
10進数列      := 数字 【 数字 】
16進数列      := {数字 | A | B | C | D | E | F} 【 16進数列 】
記号・文字列  := 記号・文字【記号・文字列】
レジスタ記号  := { AL | CL | DL | BL | AH | CH | DH | BH |
                  AX | CX | DX | BX | SP | BP | SI | DI |
                  ES | CS | SS | DS | FS | GS | EAX | ECX |
                  EDX | EBX | ESP | EBP | ESI | EDI | CR0 |
                  CR2 | CR3 | DR0 | DR1 | DR2 | DR3 | DR6 |
                  DR7 | TR3 | TR4 | TR5 | TR6 | TR7 }
予約語        := { $ | ? | DUP | プロセッサ命令 | ディレクティブ |
                  演算子文字列 | レジスタ記号 }

2進数         := 2進数列B
10進数        := 10進数列 【 D 】
16進数        := 16進数列H
文字列        := '文字列文字'
シンボル      := { 一般記号 | 一般文字 } 【 シンボル文字列 】
ラベル        := シンボル:
LAC値         := ドル記号
数値要素      := { 10進数 | 16進数 | 2進数 }
要素          := { (論理和式) | [論理和式] | 数値要素 | LAC値 |
                  文字列 | レジスタ記号 | シンボル }

セグメント式  := 要素 【 : 要素 】
属性式        := 【 属性演算子 】 セグメント式
符号式        := 【 { + | - } 】 属性式
乗除式        := 符号式 【 { * | / | MOD | SHL | SHR } 符号式 】
加減式        := 乗除式 【 { + | - } 乗除式 】
比較式        := 加減式 【 { GT | GE | EQ | NE | LE | LT }
                 加減式 】
論理否定式    := 【 NOT 】 比較式
論理積式      := 論理否定式 【 AND 論理否定式 】
論理和式      := 論理積式 【 { OR | XOR } 論理積式 】
結合式        := [論理和式]【 結合式 】
式            := 論理和式 【 結合式 】
ショート式    := SHORT 式

コメント      := ;【 記号・文字列 】
オペランド    := { 式 | ショート式 }
                 【 【 空白文字 】,【 空白文字 】 オペランド 】
行            := 【 空白文字 】 {【 ラベル 】【 空白文字 】
                 【 プロセッサ命令【 空白文字 オペランド 】】 |
                 【【 シンボル 空白文字 】
                 ディレクティブ【 空白文字 オペランド 】】}
                 【 空白文字 】【 コメント 】 終端記号
```


アセンブラ用語集

エミュレーション　あるシステム上で別のシステムを模倣すること．数値演算プロセッサを持たないシステムでは数値演算エミュレータを使う．

IC千一夜

第12回 ——— CRTコントローラの系譜



虎岩 登

普段，あまりコンピュータを扱う機会のない人に，コンピュータの絵を描いてもらうと，どんな絵を描くでしょうか．20年も前なら，戸棚のような大きさの箱に電球がたくさん付いたものと，オープンリールテープが目玉のように見える同じくらいの大きさの箱を並べて描いたものになることが多かったように思います．今ではもちろんほとんどの方が，キーボードと小型のテレビのそばにフロッピーディスクの取り出し口を付けたケースを描いてくれることでしょう．それくらい，コンピュータの出力装置として，ブラウン管，つまりCRTを用いたモニタを利用することが一般的になっています．今回は，そのCRTに文字や図形を出力するために使われるLSI，CRT controller，略してCRTCについて見ていきます．

半導体メモリの量産化

CRTディスプレイの普及

皆さん，CRTの付いていないコンピュータを想像できるでしょうか．CRTディスプレイがよく使われるようになったのは，マイクロコンピュータができる少し前，TTL-ICというロジックICファミリーが広く使われるようになり，それなりに高速なメモリも使えるようになってからです．

CRTというデバイスは，コンピュータの出力装置としては扱いにくいものです．その理由は，1秒間に何十回もリフレッシュ，つまり表示を繰り返えしていないと人の目には見えないからです．プリンタなら，メカニズムを駆動して紙の上に活字を押し付ければ，それ以降はコンピュータの方から何もしなくても字が残っていますし，人が読むことだってできます．ところがCRTはつねに1画面分のデータをコンピュータから転送してやるか，特別のメモリに記憶してそこから自動的にCRTへ送り出すかしなければなりません．前者ではCPUの能力のほとんどを1台の端末のために割かなければならないので，例外的な場合にしか使われません．後者の場合，メモリが安くなるまではかなり高価な回路でした．

それが1970年代に入り，半導体メモリの量産によって安価なメモリが供給されるようになると，CRTディスプレイは，プリンタのように紙やインクなどの消耗品を必要とせず，無音ですから，普及するわけです．

そのころ，大型コンピュータの端末としては，メモリ以外のほとんどすべてをTTL-ICなどの集積度の低いロ

ジックで組み上げた（制御用CPUを用いない）タイプの端末機から，じょじょにマイクロプロセッサを制御に用いた比較的高機能なCRT端末が用いられるようになります．

同時期に，今のパーソナルコンピュータの直接の先祖ともいえる，ワンボード・コンピュータにCRTディスプレイとカセットテープ・インターフェース（カセットテープを外部記憶装置として使っていたのです）を備えた，ホビー用コンピュータが出てきます．この時の一般的なCRT表示用の回路構成は図1のようなものでした．表示用のバッファメモリは，CPUからはメモリとしてアクセスできます．同時にこのメモリはTTL-ICで構成されたカウンタ群からアドレスを供給されて，データを読み出されています．読み出されたデータは，文字コードのままですから，これをフォント情報に変換するためのROMに入力し，出力されるフォント情報を直列に変換してCRTへ送り出します．カウンタ群は，同時にCRTの同期信号などのタイミングを作成するのにも使われます．

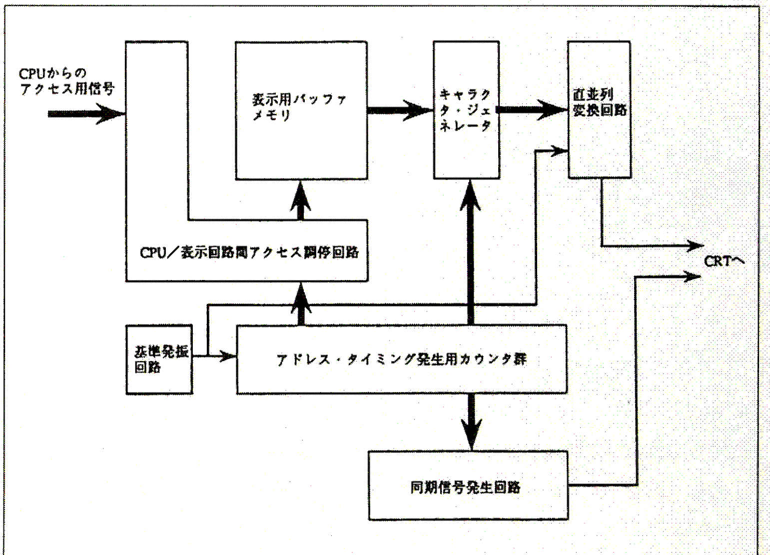
この回路は，CPUからは単なるメモリにしか見えないようになっていたので，よくVideo-RAMなどと呼ばれていました．CPUが読み書きするときには，CRTへ出力するデータが化けたりするので，画面にノイズのようなものが見えたりする回路も，このころはよく見られました．まぁ，ホビー用としては，できるだけ回路を簡単にして安くしようとしたためでしょうが，それだけ部品などのコストもかかっていたといえるかもしれません（当時の32文字×16行のV-RAM基板の値段で，今ならS3社のチップを使ったSVGA基板を買ってもおつりがくるくらい）．

キャラクタからグラフィックへ

プログラマブルLSI

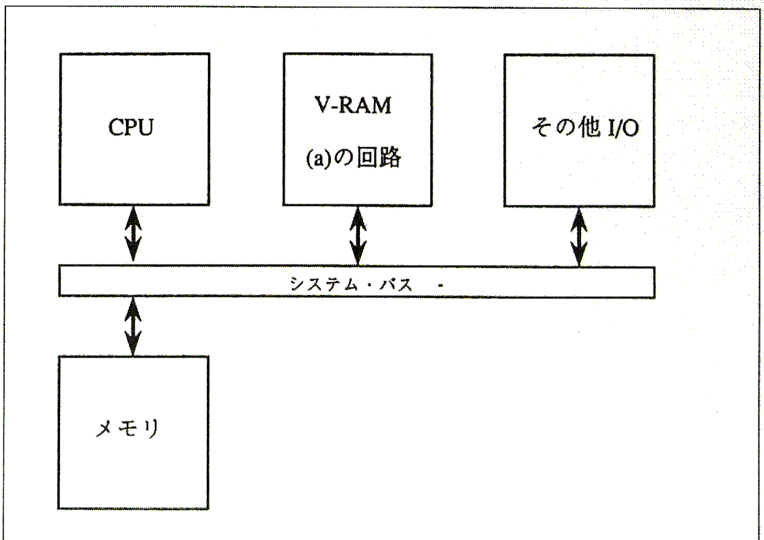
ところで，この回路構成だと，カウンタ群の設計変更によってさまざまな画面構成がとれます．逆にいえば，64文字×16行構成ならそういうカウンタ回路を，80×24行なら別のカウンタ回路を構成する必要があります．その回路設計に工夫のできた時代は，でも，そんなに長くはありませんでした．1つの画期的なLSIが作られたからです．それは，表示バッファ用のメモリをアクセスするためのカウンタ群と，同期信号を作り出すカウンタ群を内蔵して，しかもどちらも任意の数をカウントできるような，プログラマブルな

図1a　Video-RAM方式の表示回路の一般形式



初期の表示回路はメモリ以外のほとんどすべてを集積度の低いTTL回路で作成した．

図1b　Video-RAMを用いたシステム構成



V-RAMはメモリと同じようにCPUからアクセスされる．

カウンタにしたLSIでした．オリジナルは日立のHD46505です．CRTC (CRT Controller)と呼ばれていました．内部構成を図３に示します．

このLSIは1980年代前半ごろまで，たいへん多くのコンピュータや端末装置で採用されました．その理由は，内部のカウンタがプログラマブルになっていたためです．32×16文字でも64×16文字でも40×25文字でも80×24文字でも，あるいはグラフィックディスプレイでも，どの表示構成の装置に使うのにも，LSIはたった１種類でよかったのです．だからそれだけ量産も可能で，結果的に価格も安くなるという，「LSIの論理」を代表するICであったといえるでしょう．モトローラ社がセカンドソースとなりMC6845の名前で出荷していました．また，富士通からはずっと後になってMB89321/MB89322といった上位互換の改良版チップも作られています．

これと同時期に，Video-RAMとは別の考え方の文字表示用LSIがありました．NECのμPD3301です．これは，図４のようにメインメモリの一部を表示バッファ用メモリとして利用します．DMA，ダイレクト・メモリ・アクセスという手段で，定期的にCPUを一瞬止めて，データをCRTCの内部バッファへ読み込みます．この内部バッファは，表示１行分の容量をもつものが２本用意されており，ある内部バッファをCRTへ出力している最中にもう一方のバッファへDMAで転送をすることで，表示とデータ転送がぶつからないようにしています．CPUがDMAによって動作中断されている時間は数パーセントにしかならないので，ほとんど影響がないといってもよいでしょう．この方法では，DMAC (DMAコントローラ) の設定によって，メインメモリのどの場所でも表示することができるので，それなりに優れてはいます．しかしながら，キャラクタ・ディスプレイだから実用的なパフォーマンスを得られるのであって，来たるべきグラフィック・ディスプレイ時代には使えない手法でした．

キャラクタ・ディスプレイならメインメモリの表示用バッファからCRTC内部のバッファへ読み込まれるのは，文字コード（と色などのアトリビュート・コード）だけでよく，１行80文字なら80文字分の文字コードを，１行を構成する走査線(８～12本)と同じ回数くり返し内部バッファからキャラクタ・ジェネレータを介してCRTへ転送しているだけですから，意外なほどメインメモリへのアクセスは少なくて済みます．ところがグラフィック・ディスプレイだと走査線ごとに異なったデータを表示するわけですから，この方式だとつねにメインメモリにアクセスするためにCPUが動作する時間が少なくなって実用的でなくなります．このLSIはNECのPC-8001シリーズなどに採用されていましたね．

もう１つ別の方式のCRTCの系譜もあります．VDP (Video Display Processor) と呼ばれたテキサス・インスツルメント社のTMS9918です．このLSIは家庭用ゲーム機やローエンド・パソコン用に，通常のTV受像機に低解像度でカラー表示を行うために作られたものです．ハードウェア構成としては，図５のように，このLSIとD-

図２　CRTC HD46505のピン配置図

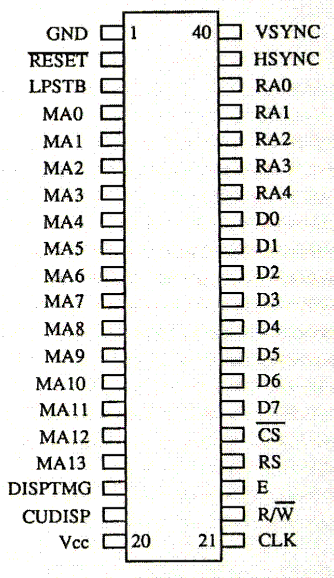
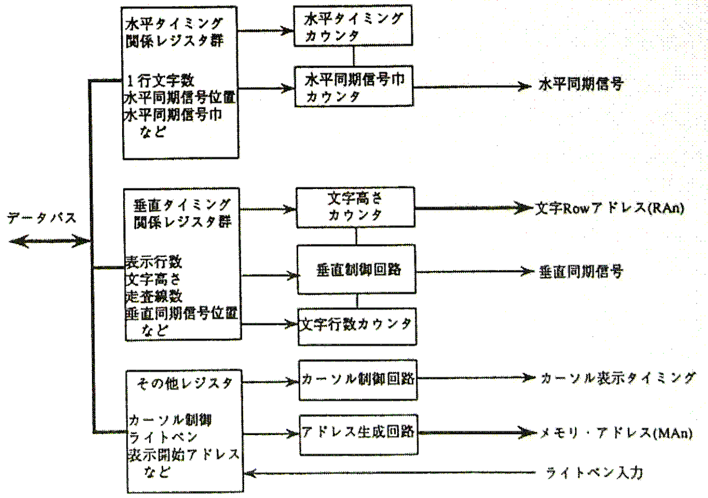


図３　CRTC（HD46505）の内部構造



タイミング規定用の多数のレジスタ群と，表示用タイミング作成用の多数のカウンタ群から構成されている．

RAMをいくつか接続するだけで，ほとんどそのまま表示部分が完成するという単純さでした．CPUからは単なるI/Oポートに見えます．その割に高機能で，グラフィック・ディスプレイにもキャラクタ・ジェネレータにもなり，キャラクタ・ディスプレイの時にはキャラクタ・ジェネレータもバッファメモリ上の一部に割り当てるために任意の形状の文字が使えますし，スプライト機能もついています．初代MSXに採用されたことから，使った記憶のある方も多いかもしれません．そのほか，あのファミコンの表示用LSIも，この改良型といってもよいと思います．

アーキテクチャ的にはHD46505と同等のVideo-RAM型ですが，用途や目的としてはこれと同様のTV表示用CRTCとして，モトローラ社のMC6847なんかもあります．こちらは，国内ではあまり採用された例がないと思いますが，RFモジュレータ用のチップとセットで，簡単にTVのアンテナ端子に接続してカラーグラフィックができるようになっていました．

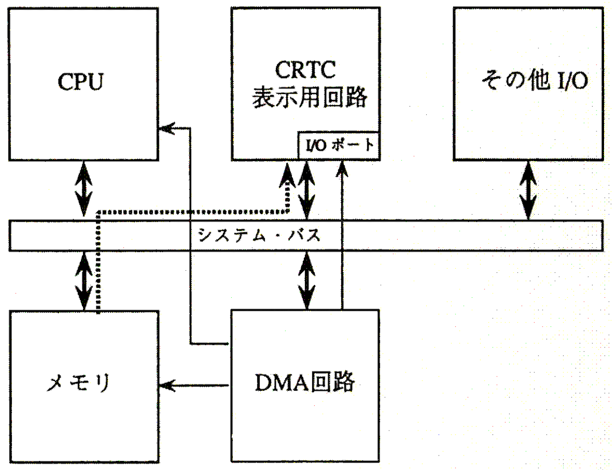
グラフィックディスプレイの時代へ

高解像度・高速表示をめざして

さて，1980年代はじめはグラフィック・ディスプレイ用の高機能LSIの要求が高まっていました．応用上はPC-9801並みのグラフィック・ディスプレイであれば，表示回路を作るだけならHD46505などのCRTCを流用すれば可能なのですが，なにせ当時はほとんどのCPUが8bitで，16bitのマイクロプロセッサは出荷されはじめてはいましたが，本格的には応用されていなかった時代です．画面に線を引くにしても，塗りつぶすにしても，とにかく描画しているところが目で追えるほど遅かったのです．そこで，描画機能までハードウェア化した，専用のグラフィック・ディスプレイ・コントローラが作られました．代表的なものが，今もPC-9801シリーズに使われているGDC，μPD7220です．

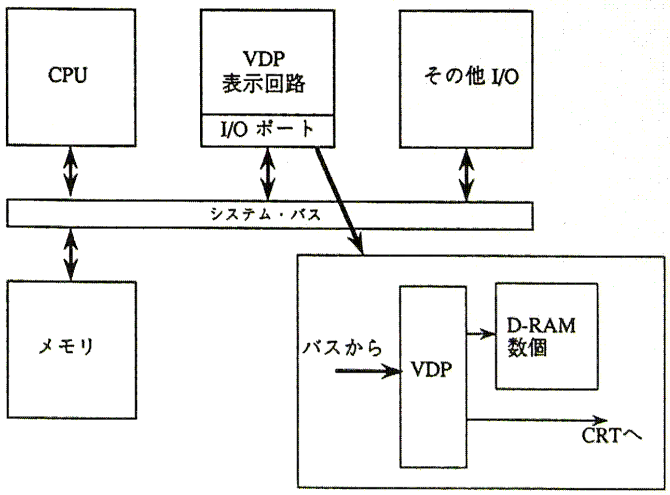
このLSIは，キャラクタ・ディスプレイのコントローラとしても使えますが，ブレゼンハムの直線描画アルゴリズムや同じくDDAを用いた円の描画アルゴリズムをハードウェア化してあり，おもにグラフィック表示・描画に使われます．アーキテクチャ的には，外付け回路が多いのですがVDPと同

図4　DMAデータ転送方式を用いたシステム構成



DMA回路は定期的にCPUを止めて点線の経路でCRTCへデータ転送を行う．
CPUは初期設定用のI/Oポートのみアクセス可能．

図5　DMAVDP（TMS9918）を用いたシステム構成



CPUからはI/Oポートとしてアクセス可能．

じくCPUからはI/Oポートとして見えるだけの，表示用バッファメモリを自分で管理するLSIです(図6)．ただ，表示用バッファメモリは，Video-RAMと同じように，画面に基本的に表示すべきデータが順番に入っているだけの構成ですから，オプションのデータ経路としてCPUからVideo-RAMのようにアクセスする回路を付加することもよく行われました（PC-9801シリーズがそうですね）．

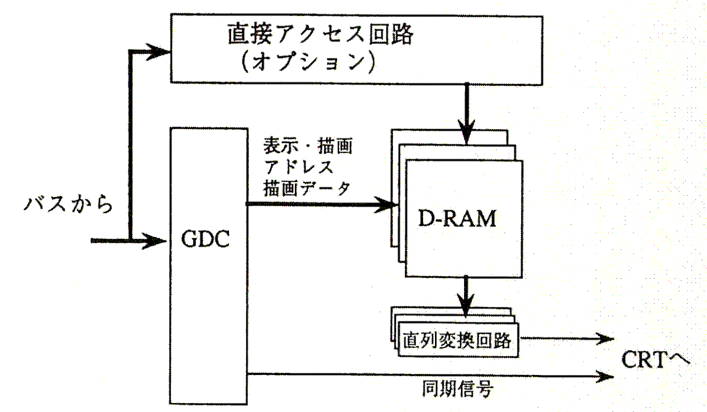
このμPD7220から遅れること数年，日立から閉領域の塗りつぶしまでを含めた非常に高機能な描画が可能なHD68464がACRTC（Advanced CRTC）と銘打って出荷されました．

これらのLSIを採用すれば，8bitや初期の16bitのCPUでもそれなりの速度でグラフィック表示が行えるようになります．しかし，だんだんこれらLSIの描画機能が活用されることが少なくなっていきます．1つはCPU自体が高性能化してしまい，CPUを使って描画処理をしたほうが高速になる場合ができたためです．もう1つは，グラフィック・ディスプレイの表示色が多くなり，CAD的な線画を多用する画面よりも，自然画に近い画像あるいは多様なコンピュータグラフィックの画像表示の応用が増えたため，LSIに内蔵された幾何学図形の描画機能の出番が少なくなったためです．そういった応用だと，多様な画像演算処理や高速なデータ転送の必要性が高まります．

そこで，今まではオーディオ分野への応用が盛んだったDSP（Digital Signal Processor）という，乗除算を含む単純な演算とデータ転送だけは得意な特殊なマイクロプロセッサに，同期信号発生回路やリフレッシュ回路などCRTディスプレイ用の機能を付加したTMS34010などが，グラフィック・アクセラレータとして使われたりしています．が，まだまだ市場が小さくいろいろなメーカーで特製のチップを開発している状態で，今後どのようなスターが現れるかわからないという状況だと思います．

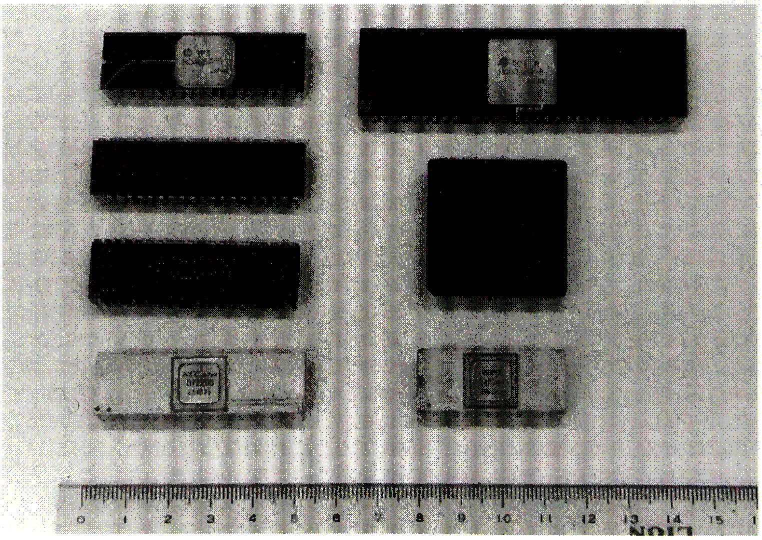
最後に，最近なにかと話題になるIBM PCについてふれておきます．初期のPC用ビデオカード，つまりCGAのころはHD46505（MC6845）を用いていました．しかしながら，高機能化をめざしてそれ以降はカスタムチップ化され，とくに近年では高解像度・高速表示をめざした専用LSIがVGAカード相当品用に多くの会社で開発されるようになっています．MS-Windowsでの性能向上をめざしたチップなど，本誌の西川氏の記事（'92年1月号）にも書かれていますね．いやはや，最近のカスタムチップの設計が容易になったことと，IBM PC関連の出荷量と競争には感心してしまいます．

図6　GDC（μPD7220）を用いたグラフィック・ディスプレイの構成



VDPとくらべて外付け回路が多いが，CPUからは同じように基本的にI/Oとしてアクセスする．

写真1　HD46505(左上)，MB89321(左中上)，TMS9918(左中下)，μPD722(左下)，HD63484(右上)，i82786（右中），D472（右下）



コンピュータのなかの

SF

志　賀　隆　生

第３回

HAPPY BIRTHDAY HAL!

この連載の初めにも書いたように，今年はコンピュータHAL9000の誕生した年にあたる．わが国でもささやかなイベントがあちこちで開かれ，それなりの関心を集めたことは記憶に新しい．なんでも，日本でのこうしたイベントの存在を耳にしたAI研究の大御所，MITのミンスキー教授は，参加できなかったことと，アメリカではその手の催し物が開かれなかったことを大いに残念がる電子メールを，とある主催者に送ったとか（これは未確認情報だが，ミンスキー自身が相当のSFファンでもあるし，お祭り好きでもあるようなので，いかにもありそうな話ではある）．

さて，HAL9000である．私たちはHAL9000と聞くと映画に登場するコンピュータを想像してしまい，アーサー・C・クラークの小説版『2001年宇宙の旅』のことは忘れがちだ．これは，映画そのものが圧倒的な迫力をもっており，小説的想像力をそれほど必要としないということもあるだろう．科学的迫真度を比べても，両者共に満点に近い．しかし，クラークの小説版が映画のノベライゼーションであるにもかかわらず，両者を比べてみるとかなりの点で食い違っている．実際，『2001年宇宙の旅』について言及するとき，どちらのバージョンをもとに話をすればよいのか迷うほどなのだ．

たとえばHAL9000の誕生日．映画では1992年１月12日，イリノイ州のコンピュータ工場ということになっているが，小説版では1997年１月12日となっていて，キューブリックが５年もさばを読んでいた（！）ことがわかる．

クラークの描くHAL9000は，「わたしは恐ろしい……怖いよデイヴ」とも言わない．小説版のHAL9000は，ボーマン船長が記憶ユニットを引き抜きはじめても冷静さを保っているようであり，あるところまでユニットを引き抜かれると，一瞬後にその思考を止める．これは私たちの多くが漠然と想像しているAIのイメージに近い．機械仕掛けの生命体に派生する〈無機的〉〈植物的〉な精神である．電源コードの付いた知性，コンセントがそばにないと生

きていけない〈たましい〉である．それは，スイッチをひねると，パチンと音がして，すべての機能を停止させる，ちょうど机の上のパソコンのように．

しかし，映画版は見悶えするように哀願を繰り返す．ボーマンの措置を何とか回避しようとする．しかし，ボーマンは機械的に作業を進めていく．徐々にHAL9000の意識が薄れていくように見える．観客の多くは，ここでHAL9000のほうに感情移入してしまう．映画の中で，HAL9000はゆっくりと最期の時を迎えているように見える，ちょうど人間の最後のように（この場面は，映画の名場面ベスト５にはぜひとも入れたいところだ）．

クラークはHAL9000の出自について，小説版『2001年宇宙の旅』のなかで，次のように書いている．

「船の六番目の乗組員は，人間ではなかったから，そんなものはいっこうに気にかけていなかった．船の頭脳であり神経系である，高度に進歩したHAL9000型コンピューターがそれであった．」

「そんなもの」とは，地球の生活，美しい地球周辺の光景から，ものすごいスピードで遠ざかっていくこと，を指している．ボーマンら乗組員は感傷的な思いに駆られるのだが，HAL9000はOKだ．彼には帰るべきふるさとは存在しない．クラークはさらに続ける．

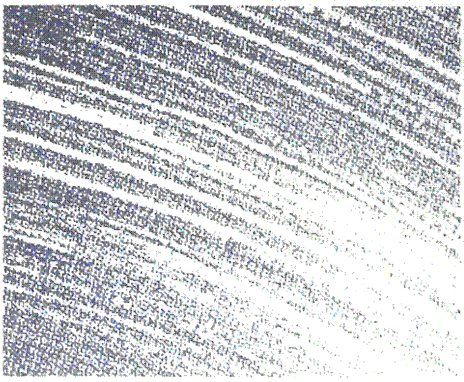
「ハル（といっても，発見能力をプログラムされたアルゴリズミック・コンピューター Heuristically-programmed Algorithmic Computerにすぎないのだが）は，第三次コンピューター革命の生みだした傑作といえた．革命は20年の期間をおいて起こるようであり，さし迫ったつぎの革命が，すでに多くの人びとの頭を悩ましていた．

第一次は，1940年代．いまでは時代遅れとなった真空管が，ENIAC以下の不器用な高速の低能児を生みだした．そして1960年代にはいると，ソリッド・ステート・マイクロエレクトロニクスが完全なかたちをとった．その科学の出現によって，ヒトが実用に供する程度の人工知性なら，会社のデスクぐらいの大きさですむことがはっきりした――もちろん，作りかたがわかればの話だが．

おそらくそこまでは，だれにもわからないだろう．だが，それは問題ではなくなった．1980年代になって，ミンスキーとグッドが，任意の学習プログラムに従って神経回路網を自動的に発生させる――自己複製させる――方法をおおやけにしたからだ．人工頭脳を，人間の頭脳の発達に酷似したプロセスで成長させることが可能になったのである．細かい仕組みについては，どの部分をとってもわからない．たとえ解明できるとしても，それは人間の理解力を百万倍も越える複雑なものだろう．

仕組みがどうあろうと，最終的な産物は，人脳の活動の大部分を，人脳よりはるかに優れた速度と確実さで再生する――哲学者のなかには，まだ「模倣する」という語を好んで使うものもいた――機械の知性であった．それははなはだしく高価につくので，今のところ完成しているHAL9000シリーズはほんの２，３台だったが，有機的な脳のほうが技術もいらず，たやすくできるだろうという使い古された冗談は，少々空ろにひびく時代となっていた．」

機械の知性は高価に付く！　ミンスキーが登場するところはニヤリとさせるが，この時代（1960年代後半）の誠実な未来予測家は，ニューラルネットワークに大きな期待をかけていたことがわかる．クラークの描くHAL9000型AIのイメージは，自己学習するニューラルネットワークの発展した姿なのだ．しかし，映画版のHAL9000は，もう少し別の方向を指し示している．それが何であるかを知るには，もう少し時間が必要である．



●『2001年宇宙の旅』（2001：A SPACE ODYSSEY，1968）アーサー・C・クラーク，伊藤典夫訳，早川書房

祐安重夫

プログラマの道具学

第18回　秋葉原

Illustration
Mikio Sasaki

私たちの周囲にある電気製品のほとんどに，現在ではマイクロプロセッサがくみこまれているのが，あたりまえになっている．普通の人はそれを意識することはあまりないだろうが，商売柄だろうか，たとえばエアコンを操作していたり電子炊飯器で飯をたいていたりするときに，内蔵されたマイクロプロセッサのソフトウェアをつくったプログラマがいることに気づいて，奇妙な感慨におそわれることがある．

マイクロプロセッサが登場する以前には，ソフトウェアの仕事のおおくは，原則としてコンピュータのハードウェアとは無関係なものだった．あるコンピュータで動作するプログラムをつくるために，そのコンピュータの内部構造や，周辺機器とのインターフェースの電気的仕様などといったことを理解する必要は，計測とか制御といった特別な分野をのぞいて，ほとんどなかったのである．

ところがマイクロプロセッサの登場によって，状況がおおきくかわってきた．コンピュータ自身がLSIチップのくみあわせで，目的に応じて自由に構成できるようになり，いままでかんがえられなかったような多彩な周辺機器が接続可能になった．その結果として必然的にプログラムを作成する際にも，ハードウェアの構成を理解する必要がでてくる．

それまでは主にコストの関係で実現できなかった分野でも，計測や制御の目的にコンピュータが使用できるようになってきた．マイクロプロセッサが存在しなければ，家庭用の電気釜や冷蔵庫にコンピュータをくみこむなどという発想は，でてこなかったにちがいない．温度や時間を計測しながら，機械の動作を制御していくという点で，現在の電気釜や冷蔵庫，電子レンジなどは，コンピュータによる計測制御システムの端的な例といえるだろう．

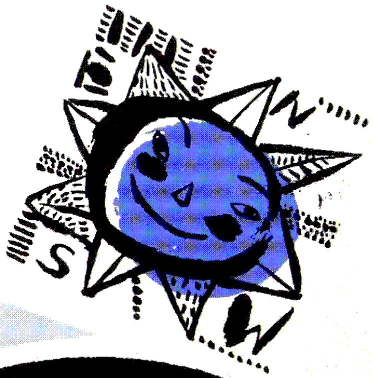
もちろん事務処理分野のアプリケーションや，大型コンピュータのプログラミングといった分野では，ハードウェアとは無関係にソフトウェアを開発しているが，現在ではハードウェアとかかわりをもたなければなりたたないソフトウェアの比率は，たぶん増加する傾向にあるとおもわれる．

私自身の仕事では，冷蔵庫の制御といった専門的に特化したものはないにしても，パーソナル・コンピュータに特殊な周辺機器を接続するためのプログラムや，特別な用途のために設計されたコンピュータのためのソフトウェアの開発は，それなりの部分をしめている．

秋葉原という街は，普通の人にとっては家電製品をかいにいくところだし，本誌の読者のようなパーソナル・コンピュータのユーザにとっては，コンピュータや周辺機器，ソフトウェアといったものをみにいくところだろう．私にとってもそれはかわらないのだが，同時にそこはLSIチップなどの電子部品や，それにかかわる情報を入手するための場所でもある．

それともうひとつ，秋葉原の周辺には東京の下町のイメージを色こくのこした地域が存在していることも，私の仕事にとっては大切な意味をもっている．秋葉原のかえりに連雀町のＹやＭで蕎麦をたべたり酒をのんでいると，仕事でたまったストレスが解消されるという効能もある．

Illustration : Masami Nozue



アルゴリズム篇 第6回
ツリー・その2

実践C塾

三代目塾頭
㈱ジャステックラボ
加野島英渡／山本博昭

今月は，先月紹介したツリー構造の弱点について解説する．また，テーマツールの単語リスト構築部分を，ツリー構造を用いて処理するように改造する．さらに，テーマツールとして作成してきた「WCOUNT.EXE」について，改造の歴史と結果をふりかえってみる．

先月までのあらすじ

データを効率よく整列させたり，検索したりというプログラミングアルゴリズムを紹介してきたこの連載だが，先月取り上げた「ツリー」は，今までと少々傾向が異なり，処理アルゴリズムというよりは，データ構造そのものに注目することによって検索の効率をよくするというものであった．

また，前回解説したように，「ツリー」のデータ構造自体は，かなり単純なものである．「自分より値の小さいデータは左，大きいデータは右に，それぞれ枝をのばして関連づけていく」．ただこれだけなのである．こうして構築されたデータ構造を検索するには，根（ルート）のデータと検索対象データの比較から処理を行っていく．対象データが根より小さい場合は，根から見て左側にしか該当データの存在する可能性がない．そこで今度は根の左側の「子」との比較を行い，その結果によってまた右または左の「子」に進むという手順である．

ある地点で2つのデータを比較し，その結果によって右に進むか左に進むかが決定するという意味では，以前取り上げた「バイナリサーチ」に似ているが，バイナリサーチが物理的に連続したデータ列をしだいに検索対象幅を狭めながら処理していくのに対し，ツリーはポインタで関連づけられた次のデータへアクセスしにいくという違いがある．ポインタで結ばれている各データは当然のことながら物理的に連続した領域に存在する必要はないので，データを追加する際もポインタをつけかえるのみで，データ列自体の移動は不要である．このため，大容量のデータを扱う場合には，バイナリサーチによるソート済みデータ列構築よりもはるかに効率のよい処理が期待できる．

今月の重点項目

ツリー構造の基本的な考え方と，おもにその利点を紹介してきたが，あまり誉めてばかりいるとツリーから何がしかの金品を受け取っているのではないかと疑いをかけられかねないので，このへんでツリー構造の暗黒面を指摘しておこう．扱うデータの性質や出現パターンによっては，ツリーは非常に効率のよくないデータ構造になってしまうという話である．

ツリー構造の弱点

ツリーデータを構築するときにぜひとも考慮に入れておかなくてはならないことがある．それはツリーデータの構築が，データの出現順になされていくために，その出現パターンによってデータ構造のバランスが左右されてしまうということである．たとえば，①から④までのデータがランダムに出現するとき，構築されるツリー構造には図1のような変種が考えられる．

結果的にはツリーを構成するデータの内容はまったく同じでも，出現パターンによってこのようにツリーの姿が変わってくるのだ．ここで注目すべきなのは，ツリーの姿によって検索の効率が変わってくるということである．盆栽についてはあまり詳しくないが，木というのものはなるべく左右にバランスよく枝が広がっているのが美しい．コンピュータにおけるツリーでも，左右にバランスよく枝が構築されていないと，データを検索するのに要する処理時間にばらつきが生じる．たとえば，図1の(4)のようになってしまうとこれは単なるシリアルにならんだデータであり，ツリー構造のメリットがなくなってしまう．

図2は，まったく同じ個数と値のデータをツリー構造にした2つの例である．データの出現順序によって，検索の効率が異なることがわかる．

ツリー構造を構築する際のデータの出現順に関して，要注意な点をまとめてみると，

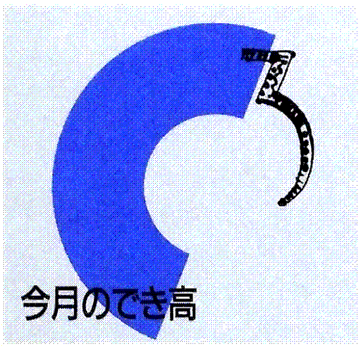
①最初のほうに出現するデータは，なるべく全データ範囲の中央付近がのぞましい．いきなり最小値・最大値に近い値が出現すると，かたよったツリーができてしまう．

②ソート済みのデータが入ってくると，ツリーは直線状にのびていくので効率が悪くなる．

とくに，いちばん最初に出現するデータはルートデータとなり，これから構築するツリー全体のバランスを決定してしまう．アルファベット順にデータをならべる処理で，いきなり'A'というデータが出現したら，ツリーはルートから右にしかのびることができなくなるわけである．

ほとんどの場合，出現するデータの順番や値が最初からすべてわかっているということはないので，ツリーの効

率は「運」次第ということになりかねないが，先ほどまとめたようにデータがソートされた状態で出現する可能性があったり，ある時点でかたよった出現のしかたをすることがある場合などには，注意すべきである．データ量が少ないときには，バイナリサーチによるデータ構築を考えたほうが効率のよいこともある．



先月号でお約束したように，テーマツールの単語リスト構築部分を，ツリー構造を用いて処理するように改造を行うことにする．連載の第1回目で仕様を決めた時点から，テーマツールのデータ構造はまったく変更していなかったので，今回の改造ははじめてのケースといえよう．

念のために，前回まで使っていたデータ構造を図3にしめす．このデータは，構造体の配列としてあらかじめデータ領域に確保してある（1000個分の配列として定義してあった）．文字列の実体を格納する領域は，これとは別にそのつどメモリ獲得して確保している．

それでは，「ツリー構造」をもった単語リストの構造を考えていくことにしよう．ツリー構造は，ポインタのみで次々とデータをアクセスできるので，いままでのように連続した領域をプログラムの中にもつ必要がない．用意するのは，ルートデータの所在を保存するポインタ変数のみである．個々のデ

図1　データ出現順序によるツリー構造の変化

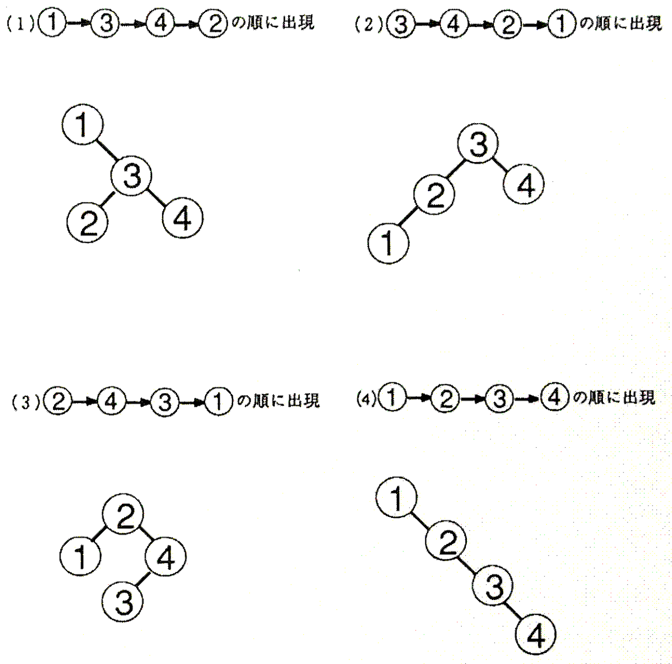
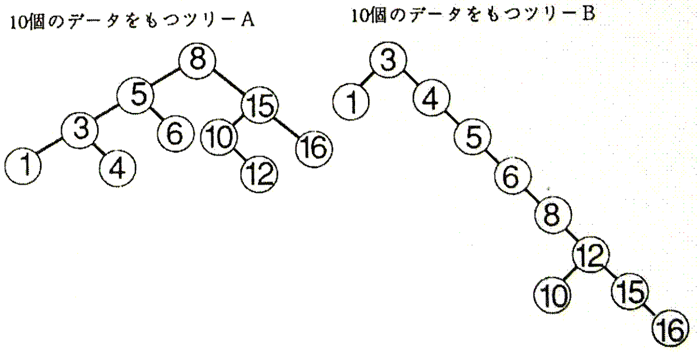


図2　同じデータ数で構造が異なるツリーを検索する



→ A，Bのツリーはまったく同じデータから構成されているが，Aのツリーでは，最悪でも4個のツリーデータをたどることによって「葉」に到達できるのに対し，Bのツリーでは7つのデータをチェックしないと到達できない「葉」がある．

ワンラインNEWS
ライオス・システムは日本IBMの「PS/55ノート」用の小型LANアダプタを発売した．リコーが発売しているLANシステムに対応している．

ータは出現するつど，メモリ獲得によって作り出す．1件のデータについて必要な情報は

①自分の左の子へのポインタ格納エリア
②自分の右の子へのポインタ格納エリア
③その単語の出現カウンタ
④その単語の文字列そのもの

以上である．文字列は単語によって長さがまちまちなので，新規に作るデータ領域は上記の①～③までのサイズ＋文字列の長さ（文字列終端のNULLまで）である．図4にこのデータ構造をまとめた．このデータ構造をCでコーディングしたものが図5である．各々のデータを格納するための構造体とルートデータのポインタを保存するための変数を用意する．それでは，このデータ構造が具体的にどう構築されていくかをデータを使って解説しよう．処理ファイルには以下の3つの単語が入っているものとする．

「Programmer」
「Shoeisha」
「JustechLabo」

このデータを読み込んでツリーを構築すると，以下のような図ができるはずである．

```
         Programmer
        /          \
JustechLabo      Shoeisha
```

プログラム上でこのツリーが育っていく動きを図6にまとめた．初出の単語が出るたびに必要な大きさのデータ領域をmalloc関数で獲得し，そのポインタを自分の親（ルートデータの場合は，変数Root）に記録していく．このツリーを検索する際には，必ずRootの示すルートデータから開始することになる．

データ構造ができたところで，いよいよテーマツールの改造となるわけだが，その前にお知らせしなければならないことがある．この連載の1回目からおつきあいいただいた「WCOUNT.EXE」は今回をもってお役御免ということになる．したがって今度の改造を終えてこのツールは一応の完成ということにしたい．そういうわけで，今

図3　従来のテーマツールのデータ構造

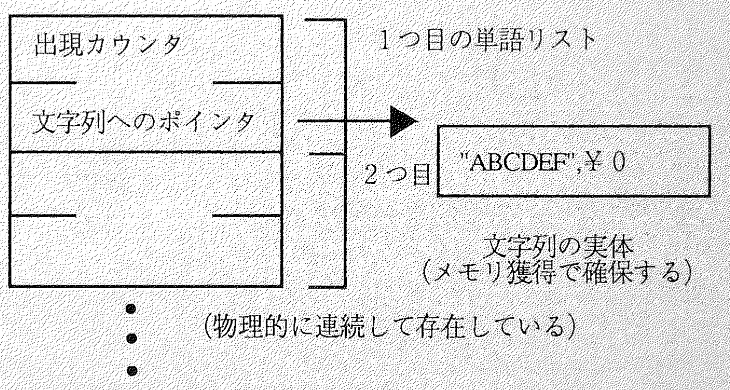
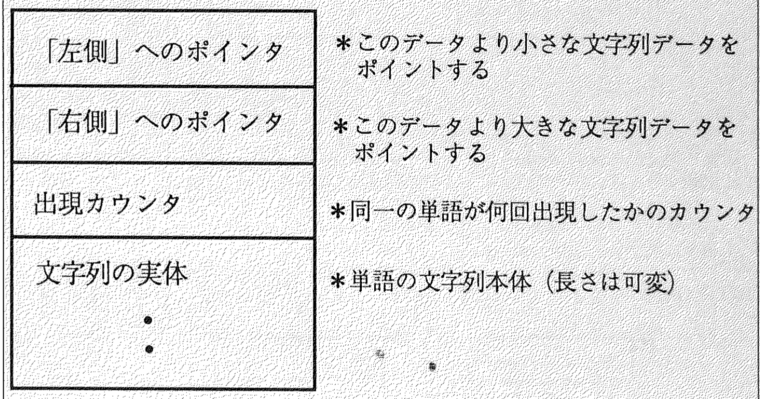


図4　単語リストのツリーデータ構造



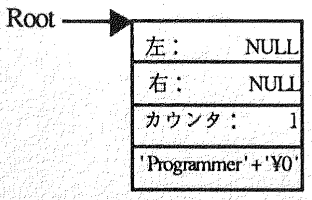
注：右側，左側に「子」がない場合には，各ポインタはNULLとする．

図5　ツリー構造型単語リストのコーディング

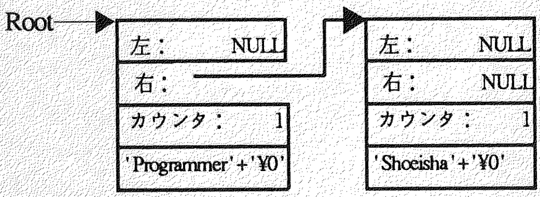
```
struct    _cell                        ※単語ごとにこのデータ構造を1つ作る
{
    struct    _cell *LeftHand;         ※左側のデータへのポインタ
    struct    _cell *RightHand;        ※右側のデータへのポインタ
    int        DataCount;              ※既出単語のカウンタ
    char    StringBuf[1];              ※文字列の先頭
};

struct    _cell    *Root = NULL;       ※ルートデータのポインタを保存しておく変数
```

図6　テーマツールの単語リストツリーが成長していく様子

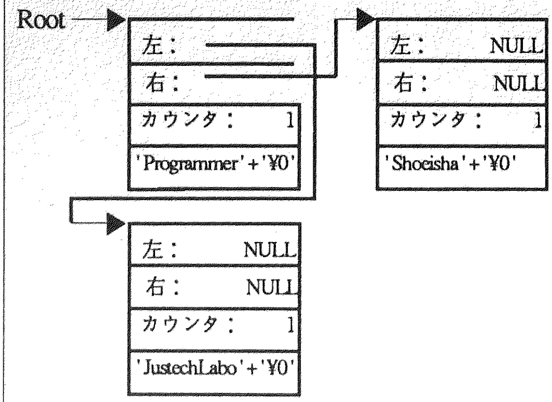
(1)はじめの単語「Programmer」を読んだとき
単語リスト用の構造体_cellのサイズ+10文字分の領域をmallocする．_cellには文字列の先頭1文字分のサイズがすでに含まれているので，文字列終端のNULLまで含んだサイズとなる．mallocが返してくる領域へのポインタをルートデータ用変数Rootに保存する．まだツリーにはルートデータのみしかデータがないため，両側のポインタはNULLになる．



(2) 2番目の単語「Shoeisha」を読んだとき
単語「Shoeisha」をルートデータ「Programmer」と比較する．この場合は右側に枝がのびるため，右側ポインタを見にいく．ポインタがNULLなので，ここに新しい「葉」を作る．単語リスト用の構造体_cellのサイズ+8文字分の領域をmallocし，そのポインタをルートデータの右側に格納する．このデータは「葉」なので，両側のポインタはNULLになる．



(3) 3番目の単語「JustechLabo」を読んだとき
単語「JustechLabo」をルートデータ「Programmer」と比較する．この場合は左側に枝がのびるため左側ポインタを見にいく．ポインタがNULLなので，ここに新しい「葉」を作る．単語リスト用の構造体_cellのサイズ+11文字分の領域をmallocし，そのポインタをルートデータの左側に格納する．このデータは「葉」なので，両側のポインタはNULLになる．



リスト1　WCOUNT.MAK

```
.AUTODEPEND

#                   *Translator Definitions*
CC = bcc +WCOUNT.CFG
TASM = TASM
TLINK = tlink

#                   *Implicit Rules*
.c.obj:
  $(CC) -c {$< }

.cpp.obj:
  $(CC) -c {$< }

#                   *List Macros*

EXE_dependencies =  ¥
  wcsub.obj ¥
  wcount.obj ¥
  a:¥bc¥lib¥wildargs.obj

#                   *Explicit Rules*
wcount.exe: wcount.cfg $(EXE_dependencies)
  $(TLINK) /x/c/P-/LA:¥BC¥LIB @&&|
c0s.obj+
wcsub.obj+
wcount.obj+
a:¥bc¥lib¥wildargs.obj
wcount
                    # no map file
emu.lib+
maths.lib+
cs.lib+
graphics.lib
|

#                   *Individual File Dependencies*
wcsub.obj: wcsub.c

wcount.obj: wcount.c

#                   *Compiler Configuration File*
wcount.cfg: wcount.mak
  copy &&|
-v
-vi
-IA:¥BC¥INCLUDE
-LA:¥BC¥LIB
-P-.C
| wcount.cfg
```

リスト2　WCOUNT.C

```
/* ======================================================================
    実践C塾・テーマツール「WCOUNT.EXE」メインソース

    WCOUNT.C            Copyright 1991,1992 bY JUSTECH-LABO. (H.kanoshima)

====================================================================== */

#include    <stdio.h>

#define     LINEMAX     1024

void    wcount_file(char *fname)
{
FILE    *fp;
static  char    readline[LINEMAX];
char    *rptr , *sptr;

    fp = fopen(fname , "r");
    if(fp == NULL)
    {
        printf("'%s' がありません¥n",fname);
        return;
    }

    printf("=== %s ===¥n",fname);

    while((rptr = fgets(readline , LINEMAX , fp)) != NULL)
    {
    extern  char    *wget(char **);
    extern  void    wset(char *);

        while((sptr = wget(&rptr)) != NULL)
            wset(sptr);
    }
}

/* ======================================================================


====================================================================== */
void        main(int argc,char **argv)
{
extern  void    dump_word(void);

    while(--argc)
        wcount_file(*++argv);
    dump_word();
}
```


ワンラインNEWS
UNIXシステム・ラボラトリーズは同社が開発したUNIXの標準的なOSである「システムVリリース4」の「ミップス版」を開発した．

月はまとめという意味からテーマツールの全ソースとMAKEファイルを掲載させていただいた．リスト2，3がそのメインソースおよびサブソースである．

実行モジュールの作成はリスト1のMAKEファイルを使って以下のように行う．

```
A> MAKE -fWCOUNT.MAK
```

サブソースの最初のほうに「#define TEST 0」という記述があるが，これはツリー構造を表示する際の経路を示すためのテストロジックの条件フラグである．ここを「#define TEST 1」としてコンパイルすることによって，単語カウント結果とともにツリー検索経路を表示させることができる．この機能の動作については，後ほど詳しく見ていくことにする．

データ構造を根本的に変更したため，サブソース「WCSUB.C」は全面的に変更を行った．この新しいサブソースの動作解説を図7にまとめる．

テスト用ロジックについて

今回の改造で単語リストデータをツリー構造としたわけだが，内部処理は大幅に変わったにもかかわらず，見かけ上の動作は今までとまったく変化ない．これでは少々ものたりないという方のために，表示の時にどのようにツリーをたどっていくかをトレースする機能を付加してみた．

前回の「TREESORT」の時にツリーの表示手順を解説したが，テーマツールも同様にルートデータからはじめて，まず「いちばん小さいもの　=　左

リスト3　WCSUB.C

```
/* ======================================================================
   実践C塾・テーマツール「WCOUNT.EXE」サブソース

   WCSUB.C         Copyright 1991,1992 bY JUSTECH-LABO. (H.kanoshima)

====================================================================== */
#include    <stdio.h>
#include    <stdlib.h>
#include    <string.h>

/*
  動作チェック用ロジックのためのフラグ。1を定義すると、ツリーをたどって
  いく過程も表示する。
*/
#define TEST    0

struct  _cell
{
    struct  _cell *LeftHand;
    struct  _cell *RightHand;
    int     DataCount;
    char    StringBuf[1];
};

struct  _cell   *Root = NULL;

/* ======================================================================


====================================================================== */
struct  _cell   *MakeCell(char *sptr)
{
struct  _cell   *CellPtr;

    CellPtr = (struct _cell *)malloc(sizeof(struct _cell) + strlen(sptr));
    if(CellPtr == NULL)
    {
        printf("メモリが足りないので単語リストが作れません¥n");
        exit(1);
    }
    CellPtr->LeftHand = CellPtr->RightHand = NULL;
    CellPtr->DataCount = 1;
    strcpy(CellPtr->StringBuf, sptr);

    return  CellPtr;
}

/* ======================================================================


====================================================================== */
void    AddTreeSub(char *sptr, struct _cell **Hand)
{
int     Judge;
struct  _cell   *CellPtr;

    if((CellPtr = *Hand) == NULL)
    {
        *Hand = MakeCell(sptr);
        putchar('.');
        return;
    }

    if((Judge = strcmp(sptr, CellPtr->StringBuf)) == 0)
    {
        CellPtr->DataCount++;
        putchar('!');
        return;
    }
    else if(Judge < 0)
        AddTreeSub(sptr, &(CellPtr->LeftHand));
    else
        AddTreeSub(sptr, &(CellPtr->RightHand));
}

void    wset(char *sptr)
{
    AddTreeSub(sptr, &Root);
}

/* ======================================================================


====================================================================== */
void    DumpTree(struct _cell **Hand)
{
struct  _cell *CellPtr;

    CellPtr = *Hand;
    if(CellPtr == NULL)
    {
#if TEST
        puts(" ↑存在せず");
#endif
        return;
    }
#if TEST
    puts("Left");
#endif
    DumpTree(&(CellPtr->LeftHand));
    printf("(%04d) - '%s'¥n", CellPtr->DataCount, CellPtr->StringBuf);
#if TEST
    puts("Right");
#endif
    DumpTree(&(CellPtr->RightHand));
#if TEST
    puts("Up");
#endif
}

/* ======================================================================


====================================================================== */
void    dump_word(void)
{
        puts("¥n");
    DumpTree(&Root);
}
/* ======================================================================


====================================================================== */
char    *wget(char **sptrptr)
{
static  char    getw[2048];
char    *sptrs , *sptrd;
int     cplen;

    sptrs = *sptrptr;

    sptrs += strspn(sptrs , " ,¥t¥n");
    if((*sptrs == '¥0') || (*sptrs == ';')) return  NULL;

    sptrd = strpbrk(sptrs , " ,¥t¥n");
    if(sptrd == NULL)   cplen = strlen(sptrs);
    else                cplen = sptrd - sptrs;
    sptrd = getw;
    while(cplen--)
        *sptrd++ = *sptrs++;
    *sptrd = '¥0';
    *sptrptr = sptrs;

    return  getw;
}
```

側に子をもたないデータ」を見つけ，その次・その次と検索していく．たとえば，下のようなデータを「SMPL.TXT」として作成して実行させた例が図8にあるが，これを使って確認してみよう．

SMPL.TXT

```
3
0
8
5
1
```

テストロジックが表示するメッセージは以下の4種類である．

「Left」
現在のデータの左側の子を見に行く

「Right」
現在のデータの右側の子を見に行く

「↑存在せず」
上記のいずれかを行おうとしたらポインタがNULLだった

「Up」
現在のデータの親を見に行く

図8の実行画面を見ながら，「SMPL.TXT」をツリー化したデータをたどっていく過程を追っていくと，次のようになるだろう．

Left　(1)ルートデータの左の子を見に行く．
Left　(2)その子は左側に子

図8　テストロジックを有効にした場合の実行画面例

```
=== SMPL.TXT ===
.....
Left
Left
 ↑存在せず
(0001) - '0'
Right
Left
 ↑存在せず
(0001) - '1'
Right
 ↑存在せず
Up
Up
(0001) - '3'
Right
Left
Left
 ↑存在せず
(0001) - '5'
Right
 ↑存在せず
Up
(0001) - '8'
Right
 ↑存在せず
Up
Up
```

図7　ツリーを使った単語リスト構築の処理概要

【ツリーに新規データを追加する】

```
: struct    _cell    *MakeCell(char *sptr)
```
追加する文字列データを受け取り，作成したツリーデータのポインタを返す．

```
: {
: struct    _cell    *CellPtr;
:
:     CellPtr = (struct _cell *)malloc(sizeof(struct _cell) + strlen(sptr));
```
これから追加するデータのために文字列長＋ポインタなどのデータ分メモリをアロケートする．

```
:     if(CellPtr == NULL)
:     {
:         printf("メモリが足りないので単語リストが作れません¥n");
:         exit(1);
```
メモリがアロケートできない場合は，これ以上処理を継続することはできないので，エラー表示を行なってプログラムを強制終了する．

```
:     }
```
メモリが確保できたら，各種データを初期化する．

```
:     CellPtr->LeftHand = CellPtr->RightHand = NULL;
```
このデータは「葉」となるため，左右の子へのポインタはNULLとする．

```
:     CellPtr->DataCount = 1;
```
新規データなので，単語カウンタは当然1になる．

```
:     strcpy(CellPtr->StringBuf, sptr);
```
文字列の実体を確保した領域にコピーする．

```
:     return    CellPtr;
```
データ領域へのポインタを返して処理終了．

```
: }
```

【指定のデータをツリーに追加する】

```
: void    AddTreeSub(char *sptr, struct _cell **Hand)
```
この関数は，検索対象の文字列と検索開始データポインタを受け取る．

```
: {
: int        Judge;
: struct    _cell    *CellPtr;
:
:     if((CellPtr = *Hand) == NULL)
```
ここがNULLになるのは以下の条件の時である．
・最初のデータ（ルートデータ）の時
・再帰検索で「葉」のデータに達した時

```
:     {
:         *Hand = MakeCell(sptr);
:         putchar('.');
:         return;
```
新規に「葉」を作成し，新出単語として「.」を表示してから処理を終了する．

```
:     }
:     if((Judge = strcmp(sptr, CellPtr->StringBuf)) == 0)
```
ポインタ上のデータが持つ文字列と検索対象文字列を比較する．

```
:     {
:         CellPtr->DataCount++;
:         putchar('!');
:         return;
```
等しい場合は，既にリスト中に存在するデータなので，カウンタを1つアップするだけ．既出の単語なので，「!」を表示して，処理を終了する．

```
:     }
:     else if(Judge < 0)
:         AddTreeSub(sptr, &(CellPtr->LeftHand));
```
小さかった場合は，左側に子がなくなるまでツリーを再帰的にたどっていき，そこにデータを新規登録する．

```
:     else
:         AddTreeSub(sptr, &(CellPtr->RightHand));
```
大きかった場合は，右側にデータがある限りツリーを再帰的にたどっていき，そこにデータを新規登録する．

```
: }
```

【ツリーデータの追加】

```
: void    wset(char *sptr)
: {
:     AddTreeSub(sptr, &Root);
```
ルートを基点に，挿入箇所を検索してツリーに追加する．

```
: }
```

【ツリーデータの表示】

```
: void    DumpTree(struct _cell **Hand)
: {
: struct    _cell *CellPtr;
:
:     CellPtr = *Hand;
```
受け取った表示開始データポインタを保存する．

```
:     if(CellPtr == NULL)    return;
```
そのポインタがNULLをさしていたら，これ以降はデータがない．

```
:     DumpTree(&(CellPtr->LeftHand));
```
再帰的に自分より左側のデータを処理する．
※この呼び出しが終了するのは，自分の左の子から下のすべてのデータを表示した時である．

```
:     printf("(%04d) - '%s'¥n", CellPtr->DataCount, CellPtr->StringBuf);
```
ポインタのさしているデータの文字列とカウンタを表示する．

```
:     DumpTree(&(CellPtr->RightHand));
```
再帰的に自分より右側のデータを処理する．

```
: }
```

【結果表示】

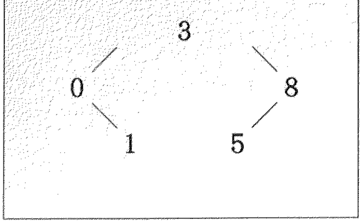
```
: void    dump_word(void)
: {
:         puts("¥n");
```
「!」や「.」を表示しているので，いったん改行する．

```
:     DumpTree(&Root);
: }
```
ルートから昇順にツリーデータを表示する．

ワンラインNEWS
ロータスはプレンゼンテーション資料を簡単に作成するソフト「フリーランス・リリース1.0J」を発売した．さまざまなデザインやレイアウトを搭載している．

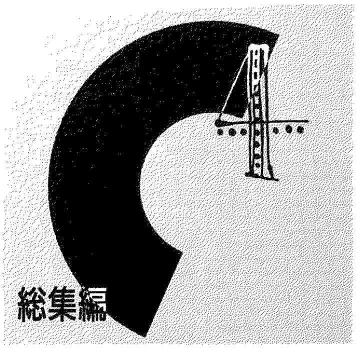
	をもっているか？
↑存在せず	(3)もっていなかった……
(0001) －'0'	(4)ということは，このデータが最小の値である．
Right	(5)このデータの右側の子を見に行く
Left	(6)右側の子は左側に子をもっているか？
↑存在せず	(7)もっていなかった……
(0001) －'1'	(8)ということは，このデータが次に小さな値である．
Right	(9)このデータの右側にまだ子があるか？
↑存在せず	(10)もっていなかった……
Up	(11)この場合は，1つツリーをあがる．
Up	(12)さらに1つあがる．
0001) －'3'	(13)このデータが次に小さな値である．
Right	(14)このデータの右側の子を見に行く．
Left	(15)右側の子はさらに左側に子をもっている．
Left	(16)その子は左側に子をもっているか？
↑存在せず	(17)もっていなかった……
(0001) －'5'	(18)ということは，このデータが次に小さな値である．
Right	(19)このデータの右側にまだ子があるか？
↑存在せず	(20)もっていなかった……
Up	(21)この場合は，1つツリーをあがる．
(0001) －'8'	(22)その親が次に小さな値である．
Right	(23)このデータの右側にまだ子があるか？
↑存在せず	(24)もっていなかった……
Up	(25)ツリーを1つあがる．
Up	(26)さらにあがろうとするが，今のはルートデータなので処理はここで終わる．

図9　SMPL.TXTのツリー図



これをツリー図にしたのが図9である．説明と照らし合わせて検証してみていただきたい．

これで，このテーマツールの改造は終了した．いつもだとここで改造の効果などを検討してみるのだが，今回は最終改造ということで，いままでの各号でのテーマツールを一堂に集めて比較することにした．以下に「総集編」としてその結果をまとめる．



この連載はこれまで6回にわたって，紹介したアルゴリズムを「WCOUNT.EXE」というテーマツールに実際に反映させながら話を進めてきたわけだが，さきほども書いたように，今回をもってこのテーマツールはお役御免ということにする．そこで今回は総集編として，「WCOUNT」の改造の歴史（?）をふりかえりながら，その改造効果などをまとめてみようと思う．6回のうち，先月号は「でき高なし」ということになってしまったので，連載で改造してきた「WCOUNT」は，表1のように5種類となる．

これら5種類の「WCOUNT.EXE」を同一のデータを使ってその効率を比較してみる．使用したデータは，Borland C++のパッケージに入っていた「DOSMAC.DOC」というテキストファイル（46722バイト）である．このファイルを「WCOUNT.EXE」で単語カウントした結果を，表2にまとめておく．

当然のことながら，5種類とも結果は同じである．

表3が各テーマツールの処理時間の一覧である．一見してわかるように，4回目と6回目のツールの圧倒的な速さと，1回目のツールの圧倒的な遅さが目立つ．そのほかにも興味深い事実があるので，箇状書きにしてみよう．

ワンラインNEWS
日本電気は電源電圧を3.3Vに抑えながらアクセス時間60nsの高速4MビットDRAMを開発，サンプル出荷を開始した．同製品は電源電圧3.3VのDRAMでは世界最速．

COLUMN　つねに一定のバランスを保つBツリー構造

本文のところで書いたツリー構造の弱点である「データの出現パターンによって，ツリーの成長に偏りが発生する」という問題点をカバーするために，いくつかの手法が考えられている．ここでは，その中から「Ｂツリー」と呼ばれるデータ構造を紹介したい．

Ｂツリーは，ツリーを構築する際に，データがどのような順番で出現してもつねに一定のバランスを保つような構造をもったツリーである．通常のツリーがデータを１つずつポインタで結んでいくのに対し，Ｂツリーではいくつかのデータの集合を１つの「葉」として扱う．各「葉」の基本的構造は図Ａのようになっている．

■Bツリーデータの検索

①のポインタがさす「葉」の中には，データＡの値以上のものは含まれていない．同様に，②のポインタがさす「葉」の中に含まれるデータは，データＡ＜ｘ＜データＢの範囲にある．このことから検索のロジックは，容易に察しがつくだろう．

(1)通常のツリーと同様にルートデータから検索を開始する．
(2)ルートデータに含まれるデータ（複数）と検索対象データの比較を行う．
(3)等しいものがなかった場合は，該当するポインタを選択し，その先の「葉」を（2）と同様の手順で検索する．該当するポインタがNULLであった場合は，検索失敗ということになる．

図Ｂを使って「９」を検索する実例をあげてみる．

(1)まず根のデータをチェックする．根には「15」しかデータがないので，左側のポインタをたどる．
(2)この「葉」に含まれているデータは「８」と「12」なので，まん中のポインタをたどって先にすすむ．
(3)次の「葉」のデータは「10」と「11」である．10の左側のポインタを見るとNULLになっているので，「９」はこのツリーに含まれていないことがわかる．

■Bツリーデータの追加

Ｂツリー構造の特徴である「バランスを保持しながらツリーをのばしていく」ために，データの追加は以下のように行う．

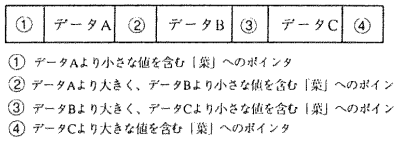
(1)データの追加位置を検索する．
(2)追加すべき「葉」の該当位置にそのデータを挿入する．
(3)この結果，「葉」の最大データ数をこえた場合は，その「葉」の中央のデータを残し，左右のデータを独立した「葉」として分割する．
(4)中央のデータは，今までの親の葉に挿入する．その結果，親の葉が最大データ数をこえた場合は，(3)と同様の手順で分割する．このようにして，分割が発生しなくなるまで上にさかのぼって処理を続ける．

分割が根まで達した場合は，いままでの根が２つに分かれ，そのまん中のデータが新しい根となる．この時，ツリーの深さが一段高くなる．

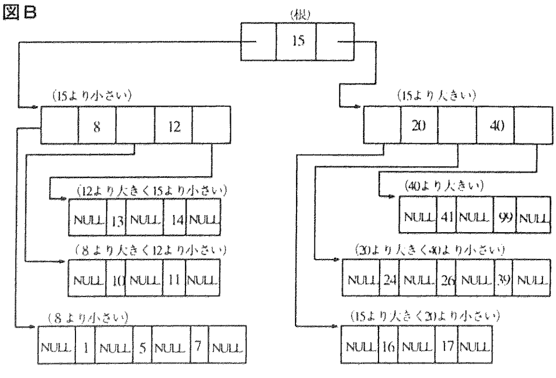
以上がＢツリーの概要である．実際にプログラミングする場合には，データとポインタのほかに，その「葉」が現在いくつのデータを含んでいるかという情報や，同一データのカウンタなどが必要になろう．

Ｂツリーの手法は，ディスク上にある巨大なデータ群から，「葉」１つずつをメモリ上に読み出して検索し，ポインタをたどって次のデータを読み込むというように大規模なデータベースの検索などに利用できる．その場合は，１つの「葉」をどのくらいの大きさに設計するかがＢツリー構造の利点を生かす鍵となるだろう．

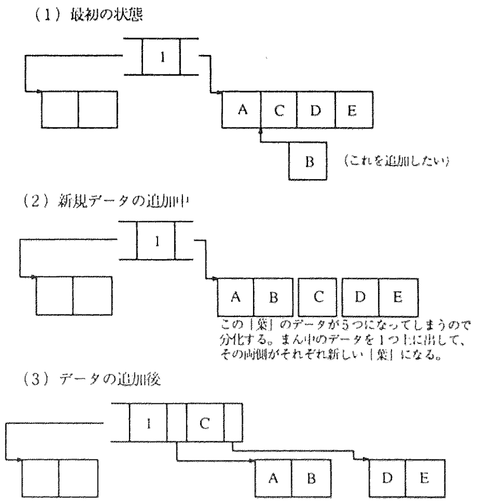
図Ａ



図Ｂ



図Ｃ



ワンラインNEWS
インテルジャパンは32ビットのRISC型MPU「i960」を拡販する．同MPUはプリンタ制御用で，とくに日本語ページプリンタに採用されている．

・1回目と2回目の44秒という差は，「バブルソート」と「クイックソート」の差そのものである．
・2回目と3回目では，同じシリアルサーチを使っているが，3回目のほうが「番人」や「学習機能」を取り入れた分だけ速くなっている．リストが完成してからの表示部分は同じクイックソートアルゴリズムなので当然同じである．
・表示時のソートが不要になったのだから，もう少し差があるように思っていたにもかかわらず，3回目と4回目で，リストが完成してから表示が終了するまでの時間差が50秒：47秒とそれほど大きくない．これは，クイックソートが高速であることと，処理のほとんどが画面表示という低速処理に費やされているためであろう．
・4回目（バイナリサーチ）と6回目（ツリー構造）の処理速度が同じである．

今回のデータ処理はたかだか1000件たらずの単語リストということで，バイナリサーチの弱点としてあげたデータの移動が，秒単位の違いが出るほど影響しないということになった．10万件とか100万件とかいったオーダーになれば，数字で違いが出てくることになろう．

なお，実験に使用した機種は，諸般の事情で「V30／10MHz」という今日となってはめずらしい部類のマシンである．最新鋭機種を使えば，この速度の何倍ものスピードが出てくるに違いない．

連載後記

6回にわたって「ファイル中の単語の出現数をカウントしてコード順に表示する」というツールをいろいろなアルゴリズムを使いながら改造してきた．そして，この過程でソート／サーチ／ツリーというプログラムアルゴリズムといえば必ずといっていいほど言及される「有名どころ」を考察し，その効果のほどをご覧いただいた．このテーマツールは今回で区切りをつけ，次回からはまた別の仕様のツールを新テーマツールとして，さらにさまざまな分野のアルゴリズムを取り上げていきたいと思う．新テーマツールは今回のものと違ったジャンルで，少しは実用になるもの，そしてプログラミングの参考になるものをということで，もっか鋭意開発中であるので，どうかご期待いただきたい．

表1　テーマツールの変遷

	単語検索ロジック	表示時のソートロジック
第1回のテーマツール	単純なシリアルサーチ	バブルソート
第2回のテーマツール	単純なシリアルサーチ	クイックソート
第3回のテーマツール	番人／学習付きシリアルサーチ	クイックソート
第4回のテーマツール	バイナリサーチ	なし
第6回のテーマツール	ツリー	なし

表2　「DOSMAC.DOC」に含まれていた単語の分布状態

出現回数	1	:	580個
出現回数	2	:	230個
出現回数	3	:	34個
出現回数	4	:	32個
出現回数	5 － 10	:	37個
出現回数	11 － 33	:	42個
出現回数	175	:	1個
出現回数	246	:	1個
総単語数			**957個**

表3　各テーマツールの処理時間一覧

	リスト完成	表示	合計
1回目のWCOUNT	1分11秒	1分34秒	2分45秒
2回目のWCOUNT	1分11秒	0分50秒	2分01秒
3回目のWCOUNT	1分07秒	0分50秒	1分57秒
4回目のWCOUNT	0分16秒	0分47秒	1分03秒
6回目のWCOUNT	0分16秒	0分47秒	1分03秒

ワンラインNEWS
日本IBMはパソコンの画面上でテレビやビデオの映像を見られる「ULTIMEDIA TV」を発売した．同装置を接続するだけでパソコンの画面の一部に映像を表示できる．

最終回

マイナーチェンジ編

酒井法雄

QuickBASICを使ってイカすプログラミングをしようというこの連載も，いよいよ最終回です．今回は，前回に作成したデータベースプログラムを改良し，構造化プログラミングの利点を再確認しましょう．

新型RX-7

RX-7がサバンナRX-7からアンフィニRX-7に変わったのは，この連載が始まったころでした．早いもので，イカすプログラミングへの道しるべとなるべく書き始めたこの連載も，いよいよ最終回を迎えました．そして，街で新型RX-7とすれちがうこともめずらしくなくなりました．

マツダのRX-7は「スポーツカー」を前面に押し出して発売されたわが国初めての量産車です．今までに2回の大幅なモデルチェンジを受けてきたRX-7ですが，その基本であるフロントミッドシップに積まれたロータリーエンジンとリア駆動の形式は変更されていません．フロントミッドシップとは，フロントタイヤと運転席の間にエンジンを積んでいるということです．この形式はコンパクトなロータリーエンジンならではの特質を生かしたものです．また，リア駆動こそスポーツカーの真髄であるという発想も多分にマニアックで魅力的なものです．

マツダの技術陣は，純粋にスポーツカーを作ることを目的としてこのクルマを開発してきたそうです．軽量コンパクトなライトウェイトスポーツだった初期型，少々大柄になったものの優れた運動性能と動力性能とスポーツカーらしいスタイリングで人気の高かった2代目，そして絶対パワーよりも軽量化を選び，卓越した運動性能を誇る最新型のRX-7．いつの時代でも日本が世界に誇るスポーツカーとしてのあり方を追及してきたその姿勢には，恐れ入るばかりです．

同じマツダのユーノスロードスターも素敵なスポーツカーですが，「人馬一体」だの，開発キャンプを「リバーサイドホテルと呼んだ」などという話を聴くと，百年の恋も醒める思いがします．純粋にスポーツカーを追及してきたRX-7のスタッフの真摯な態度のほうが私には好感が持てます．

このようなスタッフの作ってきたRX-7ですから，2回の大幅なモデルチェンジ以外にもマイナーチェンジや細かい改良が行われ続けてきました．そして，アンフィニと呼ばれる限定バージョンが折りを見て発売されました．クルマの限定バージョンというと，装備が違うとか色が違うとかの程度なのが普通ですが，アンフィニはツーシータの上，ギアレシオやスプリングやサスペンションまでもチューンした，まさにメーカー純正チューニングカーでした．そして，そのチューニングが次期マイナーチェンジに生かされました．RX-7は，まさに進化し続けるスポーツカーだったのです．

もっとも，クルマではめずらしいこのような進化も，より趣味性の高いオートバイの世界では日常茶飯事です．こちらのほうは，すでにレーサーをデチューンしたにすぎないようなレーサーレプリカが大手を振って走っています．レーサーと同時開発ですから，毎

ワンラインNEWS
日本IBMは同社製のノート型パソコン購入者が一定の料金を支払えば，紛失時などに代替機を貸し出すサービスを開始する．ノート型パソコン販売促進が狙い．

年のようにモデルチェンジします．最新型を買ってもあっという間に旧型になってしまうほどです．

さて，先月作成したプログラムは，「とりあえず動くモノ」でした．「もしかしたら，まだバグが潜んでいるかもしれません．また，使いにくいところもあるでしょう．無駄なところもあります．」と述べましたが，まさにそのとおりで，いざ吟味して使ってみるとバグはあるわ，使いにくいわ，無駄なところはあるわでした．予定通り今回は，シェイプアップし，さらに機能も拡張しましょう．

イカすプログラミングには，スポーツカーを作る絶え間ない努力と同じように，気になったところは改良し続ける努力が必要です．いくら念入りに準備をして作ったプログラムでも，一発で完璧なプログラムなどできることはまずありません．徐々にカッコよくしていけばいいのです．

バグの修正

どこかにあると思ってはいましたが，やっぱりあったのがバグです．このプログラムでは，削除メニューが正常に動作しませんでした．データを削除して，さらにそのデータを削除しようとすると，何回でもできてしまいます．これは，データを削除するところがおかしいか，ないはずのデータを検索できるかのいずれかです．

DeleteProcプロシージャを見てみると，ありましたありました．前回にも説明したとおり，このプログラムではデータの削除は物理的に削除しているのではなく，インデックスのユーザー定義型変数に削除マークをつけているだけです．この部分は次のようになっています．

```
keytab (maxindex).active =
NO
```

しかし，これでは必ずインデックスの最終レコードが削除されたことにされてしまいます．したがって，正常に目的のレコードを削除できないわけです．この部分を次のように書き直せばOKです．

```
keytab (index).active = NO
```

実につまらない間違いです．なぜこのような間違いが起きたかというと，DeleteProcプロシージャは別のプロシージャをコピーして書き直したものだからです．このとき，うっかりkeytab配列の添字をmaxindexのままにしてしまったわけです．ばかばかしい間違いですが，これは実際によくあるバグの原因です．笑うなかれ，明日は我が身です．十分に気を付けましょう．

キー消去のタイミングの変更

何度か動かしてみると，使い勝手の悪いところも見えてきます．

このプログラムでは，キーとなる名前を入力してデータを検索するわけですが，たとえば登録のときにすでにデータがあったときには，そのデータを表示し，「登録済み」メッセージを表示して再びキー入力待ちになります．このとき，前回のキーが表示されたままになっていて，その上に新しいキーを入力することになります．たとえば，「ショパン」と表示された上に「バッハ」を入力すると，みかけは「バッハン」になってしまいます．

これはどうにもカッコ悪いことになってしまいました．これを解決するには，次のような方法が考えられます．

①キーを表示するだけのフィールドを追加する
②キー入力ルーチンを改造する
③データのみでキーは消去する
④キーを含めたデータを消去するタイミングを変える

①の方法はなかなかスマートです．しかしほかのフィールドなどを含めて画面構成を考え直さないといけません．もっと簡単な方法を考えたほうがよさそうです．

②の方法は，INKEY$などを使ってキー入力ルーチンを作り，１文字でも入力されたら前の文字列を消去しようということです．しかし，このようなルーチンを作るのはかなり面倒なことになります．

③の方法はカンタンです．しかし，既存のデータとともにキーを表示しなくては意味がありませんから，キーだけ表示しないわけにもいきません．

④の方法は一番現実的な方法に思えますが，それならば「確認」くらい表示して１文字入力待ちにするくらいの気の効いたことはしたいところです．

というわけで，ここでは４番目の方法を採用しました．

まず，EraseDataプロシージャに

```
Display NX + NW, NY + 0,
WHITE, SPACE$(79 - NX - NW)
```

の１行を追加します．これで，キーも消去されるようになりました．

次に，EraseDataプロシージャを呼

び出すタイミングを変更します．従来はキー入力された直後のループ内の頭で呼び出していましたが，ループでの一通りの処理をした後に持っていきます．このプロシージャは，リスト1のようになります．これで，既存のデータがあろうとなかろうと消去されるようになりました．

しかし，このままでは，既存データが表示されてもすぐに消されてしまいます．既存データが表示されるときには必ずエラーが表示されますから，エラー時には「確認」を表示して一時停止するようにします．DispErrorプロシージャに次の行を追加します．

```
COLOR RED + REVERSE
PRINT "確認"
BEEP
dum = GetChar$
EraseError
```

また，これにともなってEraseErrorプロシージャにも，次の行を追加します．こうしないと，消したつもりが赤のリバースの帯が表示されてしまうからです．

```
COLOR WHITE
```

したがって，各プロシージャはリスト2のようになります．

無駄なゼイ肉をとる

リストをざっと眺めてみると，同じようなことを繰り返しているところがあります．たとえば，EraseDataプロシージャなどは，どう考えてもFOR～NEXTでループを作ったほうがスマートです．ただ，よく考えてみると画面設計上，連続行があるためにたまたまそうなっているからにすぎません．とはいえ，これではカッコ悪いのでリスト3のように書き直してみました．

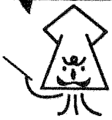
しかし，同じように並んでいるDispDataやInputDataは，同様に書き直せそうで書き直せません．これは，ユーザー定義型変数のアイテム名が違うからです．ちょっと残念ですがしかたありません．

ほかに，同じような繰り返しはないでしょうか？　探してみるとありました．メニューを表示するDispMenuプロシージャです．これもメニューの文字列の内容が違うので，一見するとこのように書かなくてはいけない気がしますが，実はメニューの文字列自体が配列menuname () に入っています．ですから，これはきれいに書き直すことができます．このルーチンはリスト4のように書き直すことができます．

また，タイトルの「データベース」という文字列は，各ルーチンにいちいち書いてあります．このような部分もまとめたほうがスマートです．次のよ

LIST1

```
SUB ResistProc
    DIM recdata AS RECSTRUCT                 レコード変数
    DIM inkeycode AS STRING * 20
    DIM index AS INTEGER
    DIM ret AS INTEGER
    DIM touch AS INTEGER

    touch = NO
    OPEN DATAFILE FOR RANDOM AS #2 LEN = 162
    DispScreen RESIST
    DO UNTIL GetKeyCode%(inkeycode) = FALSE
        EraseBuffer recdata
        EraseData  ->削除
        EraseError ->削除
        IF SearchKey(inkeycode, index, YES) = FALSE THEN
            DO
                EraseMessages
                InputData recdata
                ret = YNC%("登録しますか")
            LOOP WHILE ret = NO
            IF ret = YES THEN
                maxindex = maxindex + 1
                keytab(maxindex).customer = inkeycode
                keytab(maxindex).active = YES
                recdata.customer = inkeycode
                PutData maxindex, recdata
                touch = YES
            END IF
        ELSE
            GetData index, recdata
            DispData recdata
            DispError "登録済です。"
        END IF
        EraseMessages
        EraseData        '<-新しい位置
    LOOP
    IF touch = YES THEN
        WriteIndex
    END IF
    CLOSE #2
END SUB
```



LIST2

```
SUB DispError (str AS STRING)
    DIM dum AS STRING

    LOCATE ERY, ERX
    COLOR RED
    PRINT str; " ";
    COLOR RED + REVERSE '<-追加
    PRINT "確認"        '<-追加
    BEEP
    dum = GetChar$      '<-追加
    EraseError          '<-追加
END SUB

SUB EraseError
    LOCATE ERY, ERX
    COLOR WHITE         '<-追加
    PRINT SPC(78 - ERX);
END SUB
```

ワンラインNEWS
横河・ヒューレット・パッカードは高速でデータの読み書きができる外部記憶装置「HPディスク・アレイ」を発売した．米HPのミニコンおよびWSに接続できる．

うに定数として宣言して使うことにしましょう．

CONST TITLE="データベース"

したがって，DispMenuプロシージャも，リスト5のように書き直すことができます．

機能を拡張しよう

どうやら一通りのシェイプアップは済みました．欲をいえば，まだまだ修正したいところもあるのですが，それはみなさんのお楽しみとしてとっておくことにして，とりあえずはこれくらいにしておきましょう．

さて，次のステップとして，新しい機能を追加してみます．

このプログラムでの削除は，削除マークをつけるだけでした．なぜこのような仕様になっているかといえば，インデックスはメモリに常駐していますから，削除されたレコードを詰めることもカンタンです．しかし，データはディスク上にあるのですから，いらなくなったレコードを削除してその後のレコードを物理的に詰めてしまうには時間がかかりますから，そうカンタンにはできません．そこで，「ここは削除したんだよ」というマークをつけておくわけです．

この方法のもうひとつの利点は，削除マークを元に戻せば，データを復活することができることです．また，物理的に削除したいときには，まとめて作業すればいいわけです．

というわけで，このデータベースプログラムに，「復活」と「物理削除」のメニューを追加しましょう．

メニュー項目の追加

まずは，メニューに2つの項目を追加して，それぞれのプロシージャを呼び出せるようにしましょう．ここでは，プロシージャの中身は後から作ります．

メニュー項目は，定数と変数で内容が定義されていましたから，新しい項目を追加します．定数は，次のようにRECOVERとPDELETEを追加し，QUITの番号を一番大きくなるように変更します．

```
CONST RECOVER = 5
CONST PDELETE = 6
CONST QUIT = 7
```

また，変数も次のように追加します．

```
menuname (RECOVER) = "復活"
menuname (PDELETE) = "物理削除"
```

次に，Menuプロシージャに次の行を追加します．

```
CASE RECOVER
    RecoverProc
CASE PDELETE
    PdeleteProc
```

これで，中身のないRecoverProcとPdeleteProcを作っておけば，きちんとメニュー表示されて呼び出されるようになります．実は，さきほどDispMenuプロシージャを書き直しておいたので，このようにカンタンに変更が

LIST3

```
SUB EraseData
    DIM i AS INTEGER

    Display NX + NW, NY + 0, WHITE, SPACE$(79 - NX - NW)
    FOR i = 2 TO 8
        Display NX + NW, NY + i, WHITE, SPACE$(79 - NX - NW)
    NEXT i
END SUB
```

LIST4

```
・旧型
    CLS
    Display MX, MY, CYAN, "*** データベース (メニュー)***"
    Display MX, MY + 1 + RESIST, CYAN, STR$(RESIST) + ". 登録"
    Display MX, MY + 1 + MODIFY, CYAN, STR$(MODIFY) + ". 修正"
    Display MX, MY + 1 + DELETE, CYAN, STR$(DELETE) + ". 削除"
    Display MX, MY + 1 + SEARCH, CYAN, STR$(SEARCH) + ". 検索"
    Display MX, MY + 1 + QUIT, CYAN, STR$(QUIT) + ". 終了"

・新型
    DIM i AS INTEGER

    CLS
    Display MX, MY, CYAN, "*** データベース (メニュー)***"
    FOR i = 1 TO QUIT
        Display MX, MY + 1 + i, CYAN, STR$(i) + "." + menuname(i)
    NEXT i
```

LIST5

```
SUB DispMenu
    DIM i AS INTEGER

    CLS
    Display MX, MY, CYAN, "*** " + TITLE + " (メニュー)***"
    FOR i = 1 TO QUIT
        Display MX, MY + 1 + i, CYAN, STR$(i) + "." + menuname(i)
    NEXT i
    Display MX, MY + 3 + QUIT, CYAN, "選択してください :"
END SUB
```

できたのです．もし，DispMenuが以前のままならば，その部分も追加と変更が必要になります．このように最低限の変更で，仕様変更に応えられるようにすることも，イカすプログラミングには重要なことです．

復活プロシージャの作成

いよいよ，新しいプロシージャを作っていきましょう．例によって構造化日本語で考えてみます．

実は，このプロシージャは削除プロシージャによく似た作業をすることになります．削除プロシージャでは，検索したキーが存在し，かつ削除マークがついていなければ削除マークをつけますが，復活プロシージャでは，検索したキーが存在し，かつ削除マークがついていれば，削除マークを外すわけです．したがって，削除プロシージャをお手本にして考えたのが，図1の構造化日本語です．

ここで，削除マークがついているかどうかは，SearchKey%プロシージャが勝手に判断してしまいます．そこで，このプロシージャでは，削除マークがついたキー専用に新しいプロシージャを作るか，SearchKey%プロシージャの仕様を変更するかのいずれかの選択が必要になります．新しいプロシージャを作ること自体はカンタンですが，後から検索アルゴリズムを変更したときにも，2つのプロシージャを変更しなくてはいけなくなります．そこで，今回はSearchKey%プロシージャの仕様を変更することにしましょう．

具体的には，SearchKey%プロシージャの引数に，「act」という整数型変数を追加します．この変数に「YES」か「NO」の定数を渡すことで，削除されていないキーか削除されたキーかを判断するようにしましょう．リスト6が新しい仕様のSearchKey%です．IF文での2つ目の比較は，元は「act」ではなく「YES」になっていたものです．

これにともない，SearchKey%プロシージャを呼び出していた部分は，図2のように変更します．

DeleteProcプロシージャを改造して作ったRecoverProcプロシージャは，リスト7の通りです．

物理削除プロシージャの作成

最後に，物理削除プロシージャを作りましょう．

このプロシージャでやるべきことをまとめると，次のようになります．まず，データファイルと同じ構造のテンポラリファイルを新規にオープンします．インデックスを読みながら，削除マークのついていないレコードだけをデータファイルからテンポラリファイルにコピーします．コピーが終ったら，

▼図1　復活プロシージャ

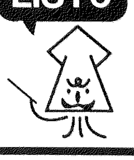
```
SUB RecoverProc
    データファイルのオープン
    画面表示
    DO UNTIL キーが入力されるまで
        バッファのクリア
        IF 削除レコードならば
            メッセージ消去
            データ読み込み
            データの表示
            「データを復活しますか」表示と入力
            IF YESなら
                インデックスのactiveをYESにする
                touchをYESにする
            END IF
        ELSE
            「データがありません」表示
        END IF
        メッセージ消去
        データ表示消去
    LOOP
    IF touchがYESなら
        インデックスを書き込む
    END IF
    データファイルをクローズする
END SUB
```

▼図2

```
IF SearchKey(inkeycode, index) = TRUE THEN
                    ↓
IF SearchKey(inkeycode, index, YES) = TRUE THEN
```

LIST6

```
FUNCTION SearchKey% (str AS STRING, index AS INTEGER, act AS INTEGER)

    DIM i AS INTEGER

    FOR i = 1 TO maxindex
        IF RTRIM$(LTRIM$(str)) = RTRIM$(LTRIM$(keytab(i).customer)) AND keytab(i).
active = act THEN
            index = i
            SearchKey% = TRUE
            EXIT FUNCTION
        END IF
    NEXT i
    SearchKey% = FALSE
END FUNCTION
```



ワンラインNEWS
東京エレクトロンは半導体の販売事業を拡販する．米VLSIテクノロジーが開発したパソコンチップセットなどの営業を本格化させ，オリジナル製品の販売にも注力する．

データファイルを削除し，テンポラリファイルの名前をデータファイルにします．もちろん，インデックスも削除されたレコードの分は詰めなくてはいけません．あまり難しくはなさそうですね．

この作業を構造化日本語にしたのが，図3です．残念ながらこの作業に似たことをしているプロシージャはないので，新たに作らなくてはなりませんが，順を追っていけばさほど難しくはないでしょう．

リスト8にこのプロシージャを示します．このプロシージャは，バグがあると一瞬にしてデータをすべて壊してしまう恐れがあります．デバッグ時には，データのバックアップをお忘れなく．

さらにイカしたプログラミングにするために

まずは，一通りのシェイプアップは終りました．使っていけば，まだまだ不満な箇所も見えてきますが，このように細かい修正を加えたり，新しい機能を追加していけば，だんだんと完璧なプログラムへと近付いていくことができます．

とはいえ，大幅な変更をしたいときには，ときとしてまったく新たに作り直したほうがよい結果を招くこともあります．そんなときにも対応できるように，汎用のプロシージャを作っておきましょう．それは新しいプログラムにも生かすことができるのですから．

ここまでは，私が作ったプログラムを通して，みなさんにイカすプログラミングのエッセンスを紹介してきました．もう，十分に「イカす」という意味がわかっていただけたことと思います．

ここからは，みなさんが「イカすプログラミング」をご自分の手で目指す番です．最後に，今回のプログラムをどう拡張していくかといった指針を示して，この連載を終りたいと思います．

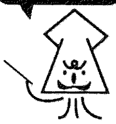
1.検索とソートアルゴリズムの探求

このプログラムで使われている検索方法は，単純に配列の内容を順番に見ていくだけです．これでは，データの件数が多くなってくるとドンドン遅くなってしまいます．そこで，より高速な検索ルーチンが欲しくなってくるでしょう．

このようなときには，データの並んでいる順序を変えてしまえば高速に検索することができます．辞書をひくときのことを考えましょう．英和辞典はアルファベット順に単語が並んでいま

```
SUB RecoverProc
    DIM recdata AS RECSTRUCT                'レコード変数
    DIM inkeycode AS STRING * 20
    DIM index AS INTEGER
    DIM ret AS INTEGER
    DIM touch AS INTEGER

    touch = NO
    OPEN DATAFILE FOR RANDOM AS #2 LEN = 162
    DispScreen RECOVER
    DO UNTIL GetKeyCode%(inkeycode) = FALSE
        EraseBuffer recdata
        IF SearchKey(inkeycode, index, NO) = TRUE THEN
            EraseMessages
            GetData index, recdata
            DispData recdata
            ret = YNC%("データを復活しますか")
            IF ret = YES THEN
                keytab(index).active = YES
                touch = YES
            END IF
        ELSE
            DispError "データがありません。"
        END IF
        EraseMessages
        EraseData
    LOOP
    IF touch = YES THEN
        WriteIndex
    END IF
    CLOSE #2
END SUB
```

LIST7



▼図3　物理削除プロシージャ

```
SUB PdeleteProc
    画面表示
    IF 「物理削除を開始しますか」がYESなら
        メッセージを消去
        テンポラリ最大変数を初期化
        データファイルをオープン
        テンポラリファイルをオープン
        FOR index = 1 TO maxindex
            IF キーがアクティブなら
                テンポラリ最大変数をインクリメント
                インデックスの添字を更新
                データを読み込む
                データを表示する
                テンポラリファイルに書き込む
                データ表示を消去
            END IF
        NEXT index
        maxindexをテンポラリ最大数にする
        インデックスを書き込む
        データファイルをクローズ
        テンポラリファイルをクローズ
        データファイルを削除
        テンポラリファイルをデータファイル名にリネーム
    END IF
END SUB
```

す．もし，これが思い付くままの順番になっていたら，辞書としての役目を果たすことができないでしょう．

これと同じように，データの追加や削除が行われたときに，インデックス順番を並べ替えるのです．もちろんそのときには，レコード番号というアイテムをインデックスのユーザー定義型に追加する必要があります．

このような並べ替えを「ソート」と呼びます．ソートにもいろいろなアルゴリズムがあります．「バブルソート」，「ヒープソート」，「クイックソート」などです．いろいろなアルゴリズムが載っている書籍がありますから参考にしてみるといいでしょう．

ソートされたデータであれば，検索も高速にすることができます．たとえば，「バイナリサーチ」という検索方法があります．辞書をひくときのことを考えましょう．本の肩に単語の頭文字を示すモノがついていない辞書であれば，次のようにひくことができます．

まず，真中を開いて目的の単語より前か後ろかを見ます．前であれば，後ろ半分の真中を開き，さらに目的の単語より前か後ろかを見ます．ここで後ろであれば，最初に開いたところと今開いたところの真中を開きます．これを繰り返していけば，少ない回数で目的の単語を見つけ出すことができます．

このように，ソートと検索のルーチンを改良すれば，大量のデータがあっても高速な検索ができるようになります．

```
SUB PdeleteProc
    DIM recdata AS RECSTRUCT                    'レコード変数
    DIM index AS INTEGER
    DIM tmpmax AS INTEGER

    DispScreen PDELETE
    IF YNC%("物理削除を開始しますか") = YES THEN
        EraseMessages
        tmpmax = 0
        OPEN DATAFILE FOR RANDOM AS #2 LEN = 162
        OPEN TMPFILE FOR RANDOM AS #3 LEN = 162
        FOR index = 1 TO maxindex
            IF keytab(index).active = YES THEN
                tmpmax = tmpmax + 1
                keytab(tmpmax) = keytab(index)
                GetData index, recdata
                DispData recdata
                PUT #3, tmpmax, recdata
                EraseData
            END IF
        NEXT index
        maxindex = tmpmax
        WriteIndex
        CLOSE #2
        CLOSE #3
        KILL DATAFILE
        NAME TMPFILE AS DATAFILE
        DispError "データの物理削除作業を終了しました"
    END IF
END SUB
```

2.ユニークでないキーに対応する

このプログラムでは，同じ名前のキーは1つだけです．「鈴木浩」という名前が2つあったとしても，「登録済みです」になってしまいます．これを避けるには，「鈴木浩2」とでもするしかありません．これはいくらなんでもあんまりです．

そこで，登録済みであっても，さらに登録できるようにしましょう．当然，検索もそれに対応して，同じキーがあったら次のレコードも見れるようにすべきでしょう．

3.あいまいなキーを検索する

現在のルーチンでは，必ずキーが全部一致していなければ検索できません．しかし，「姓は憶えているが名前はなんだったかな？」というようなこともあるでしょう．また，その逆もあるでしょう．

このようなときにも対応できるように，部分一致で検索できるようにするとよいでしょう．もっとも，キーの頭の部分一致ならともかく，真中とか後ろでも検索できるようにすると，検索速度も遅くなってしまいます．また，大量に該当データがあることも考えられます．このへんは，必要に応じてといったところでしょう．

4.複数のキーに対応する

このプログラムでは，キーは名前だけです．しかし，住所で検索したいときとか，誕生日で検索したいときもあるでしょう．このようなときには，検索したい項目だけインデックスファイルを持つという方法が考えられます．

とはいえ，全部の項目について検索したいということになれば，あまり現実的ではありません．必要最小限の項目をインデックスファイルで持って高速に検索できるようにし，あとは，遅くなりますがデータを1つずつ読んでいったほうが現実的でしょう．

5.入力ルーチンの改良

データの修正をしたいとき，このプログラムではまったく変更がなければリターンキーのみでOKですが，1つのフィールドの一部分だけを修正したいときにも全部を入力しなくてはいけません．これは，できればカーソルキーなどを使って一部だけを変更できるようにしたいところです．

現在のルーチンではINPUT文を使っていますからカンタンに済んでいますが，INPUT文を拡張したようなル

ワンラインNEWS
日本電気はDRAMの生産拠点の集約を進める．4MビットのDRAMは内外6つの拠点で生産をしているが，16MビットDRAMは広島日本電気での生産に絞る．

ーチンを作らなくてはいけないのですから，そうカンタンにはいきません．また，数値だけを入力できるとか，数値の編集ができるとか，右詰めとかセンター詰めだとかまでを指定することができるようにもしたいところです．

このルーチンは面倒ではありますが，一度作ってしまえば重宝しますから，ぜひトライしてみてください．

最後に

半年の間，おつき合いいただいてありがとうございました．つたない文章ではありましたが，何度も繰り返すことで，私の意図する「イカすプログラミング」のニュアンスはわかっていただけたことと思います（ちょっと心配ですが）．

この連載では，親しみやすいBASICで構造化プログラミングができるQuickBASICを使いましたが，紹介してきた考え方やテクニックは，ほかの言語を使ったときにも必ずや生きてくるものです．時間があれば，もう一度じっくりと読み直して，さらに自分なりのテクニックを磨いてください．そして，さらにイカすプログラミングを目指してください．この連載が少しでもそのきっかけになれば幸いです．

LIST9

db1.bas

```
DECLARE FUNCTION SearchKey% (str AS STRING, index AS INTEGER, act AS INTEGER)
DECLARE SUB RecoverProc ()
DECLARE SUB PdeleteProc ()
DECLARE SUB EraseBuffer (recdata AS ANY)
DECLARE SUB Display (x AS INTEGER, y AS INTEGER, c AS INTEGER, str AS STRING)
DECLARE SUB DispMenu ()
DECLARE SUB QuitProc ()
DECLARE SUB ModifyProc ()
DECLARE SUB DeleteProc ()
DECLARE SUB SearchProc ()
DECLARE SUB DispScreen (m AS INTEGER)
DECLARE FUNCTION GetKeyCode% (str AS STRING)
DECLARE SUB InputData (recdata AS ANY)
DECLARE SUB PutData (index AS INTEGER, recdata AS ANY)
DECLARE SUB GetData (index AS INTEGER, recdata AS ANY)
DECLARE SUB DispData (recdata AS ANY)
DECLARE SUB WriteIndex ()
DECLARE FUNCTION YNC% (str AS STRING)
DECLARE SUB ResistProc ()
DECLARE FUNCTION ReadIndex% ()
DECLARE SUB Menu ()
DECLARE SUB DispError (str AS STRING)
DECLARE SUB EraseError ()
DECLARE FUNCTION GetChar$ ()
DECLARE FUNCTION GetStr$ (x AS INTEGER, y AS INTEGER, c AS INTEGER, str AS STRIN
G)
DECLARE SUB EraseMessages ()
DECLARE SUB EraseData ()
'
'           データベースプログラム for QuickBASIC Version 1.1
'
'                          April 5 1992
'
'                     Copyright(c) 1992  Norio Sakai

'
' カラー関係定数
'
CONST BLACK = 0
CONST BLUE = 1
CONST GREEN = 2
CONST CYAN = 3
CONST RED = 4
CONST MAGENTA = 5
CONST YELLOW = 6
CONST WHITE = 7
CONST REVERSE = 8
CONST BLINCK = 16
'
' 戻り値関係定数
'
CONST FALSE = 0
CONST TRUE = -1
CONST YES = 0
CONST NO = 1
CONST CANCEL = 2
'
' 座標関係定数
'
CONST ERX = 5
CONST ERY = 23
CONST YNCX = 30
CONST YNCY = 20
CONST MX = 30
CONST MY = 2
CONST NX = 10
CONST NY = 4
CONST NW = 12
'
' メニュー関係定数
'
CONST RESIST = 1
CONST MODIFY = 2
CONST DELETE = 3
CONST SEARCH = 4
CONST RECOVER = 5
CONST PDELETE = 6
CONST QUIT = 7
CONST QUITCODE = "Menu"
CONST TITLE = "データベース"
'
' レコード関係定数
'
CONST MAXREC = 256
'
' ファイル関係定数
'
CONST INDEXFILE = "dbindex.idx"
CONST DATAFILE = "dbdata.dat"
CONST TMPFILE = "tempdata.dat"
'
' インデックス構造
'
TYPE KEYSTRUCT
    customer AS STRING * 20         '名前
    active AS INTEGER               'アクティブフラグ
END TYPE
'
' レコードデータ構造
'
```

```
TYPE RECSTRUCT
    customer AS STRING * 20      '名前
    address AS STRING * 60       '住所
    postnum AS STRING * 6        '〒番号
    phonenum AS STRING * 12      '電話番号
    age AS INTEGER               '年令
    birthday AS STRING * 10      '誕生日
    blood AS STRING * 2          '血液型
    etc AS STRING * 40           'その他
END TYPE
'
' グローバル変数宣言
'
DIM SHARED keytab(1 TO MAXREC) AS KEYSTRUCT   'キーインデックステーブル
DIM SHARED maxindex AS INTEGER                'インデックスの数
DIM SHARED menuname(RESIST TO QUIT) AS STRING 'メニュータイトル
menuname(RESIST) = "登録"
menuname(MODIFY) = "修正"
menuname(DELETE) = "削除"
menuname(SEARCH) = "検索"
menuname(RECOVER) = "復活"
menuname(PDELETE) = "物理削除"
menuname(QUIT) = "終了"

'
' メインプロシージャ
'
KEY ON
KEY 1, QUITCODE + CHR$(13)
maxindex = ReadIndex%
Menu
END

'
' 削除プロシージャ
'
SUB DeleteProc
    DIM recdata AS RECSTRUCT                  'レコード変数
    DIM inkeycode AS STRING * 20
    DIM index AS INTEGER
    DIM ret AS INTEGER
    DIM touch AS INTEGER

    touch = NO
    OPEN DATAFILE FOR RANDOM AS #2 LEN = 162
    DispScreen DELETE
    DO UNTIL GetKeyCode%(inkeycode) = FALSE
        EraseBuffer recdata
        IF SearchKey(inkeycode, index, YES) = TRUE THEN
            EraseMessages
            GetData index, recdata
            DispData recdata
            ret = YNC%("削除しますか")
            IF ret = YES THEN
                keytab(index).active = NO
                touch = YES
            END IF
        ELSE
            DispError "データがありません。"
        END IF
        EraseMessages
        EraseData
    LOOP
    IF touch = YES THEN
        WriteIndex
    END IF
    CLOSE #2
END SUB

'
' データ表示
'
SUB DispData (recdata AS RECSTRUCT)
    Display NX + NW, NY + 0, WHITE, recdata.customer
    Display NX + NW, NY + 2, WHITE, recdata.address
    Display NX + NW, NY + 3, WHITE, recdata.postnum
    Display NX + NW, NY + 4, WHITE, recdata.phonenum
    Display NX + NW, NY + 5, WHITE, MID$(STR$(recdata.age), 2)
    Display NX + NW, NY + 6, WHITE, recdata.birthday
    Display NX + NW, NY + 7, WHITE, recdata.blood
    Display NX + NW, NY + 8, WHITE, recdata.etc
END SUB

'
' エラー表示
'
SUB DispError (str AS STRING)
    DIM dum AS STRING

    LOCATE ERY, ERX
    COLOR RED
    PRINT str; " ";
    COLOR RED + REVERSE
    PRINT "確認"
    BEEP
    dum = GetChar$
    EraseError
END SUB

'
' 指定座標に文字列を表示
'
SUB Display (x AS INTEGER, y AS INTEGER, c AS INTEGER, str AS STRING)
    LOCATE y, x
    COLOR c
    PRINT str;
END SUB

'
' メニュー表示
'
SUB DispMenu
    DIM i AS INTEGER

    CLS
    Display MX, MY, CYAN, "*** " + TITLE + " (メニュー)***"
    FOR i = 1 TO QUIT
        Display MX, MY + 1 + i, CYAN, STR$(i) + "." + menuname(i)
    NEXT i
    Display MX, MY + 3 + QUIT, CYAN, "選択してください :"
END SUB

'
' データ入出力画面表示
'
SUB DispScreen (m AS INTEGER)
    CLS
    Display MX, MY, CYAN, "*** " + TITLE + "(" + menuname(m) + ")***"
    Display NX, NY, CYAN, "名　前　　: "
    Display NX, NY + 2, CYAN, "住　所　　: "
    Display NX, NY + 3, CYAN, "郵便番号 : "
    Display NX, NY + 4, CYAN, "電話番号 : "
    Display NX, NY + 5, CYAN, "年　令　　: "
    Display NX, NY + 6, CYAN, "誕生日　　: "
    Display NX, NY + 7, CYAN, "血液型　　: "
    Display NX, NY + 8, CYAN, "その他　　: "
END SUB

'
' データバッファをクリアする
'
SUB EraseBuffer (recdata AS RECSTRUCT)
    recdata.customer = ""
    recdata.address = ""
    recdata.postnum = ""
    recdata.phonenum = ""
    recdata.age = 0
    recdata.birthday = ""
    recdata.blood = ""
    recdata.etc = ""
END SUB

'
' 表示しているデータをクリアする
'
SUB EraseData
    DIM i AS INTEGER

    Display NX + NW, NY + 0, WHITE, SPACE$(79 - NX - NW)
    FOR i = 2 TO 8
        Display NX + NW, NY + i, WHITE, SPACE$(79 - NX - NW)
    NEXT i
END SUB

'
' エラー消去
'
SUB EraseError
    LOCATE ERY, ERX
    COLOR WHITE
    PRINT SPC(78 - ERX);
END SUB

'
' YNCのメッセージを消去する
'
SUB EraseMessages
    Display YNCX, YNCY, WHITE, SPACE$(79 - YNCX)
END SUB

'
' 1文字入力
'
FUNCTION GetChar$
    DIM dum AS STRING

    DO
        dum = INKEY$
    LOOP WHILE dum = ""
    GetChar$ = dum
END FUNCTION

'
' データファイルから指定データを取り出す
'
SUB GetData (index AS INTEGER, recdata AS RECSTRUCT)
```


ワンラインNEWS
マイクロソフトは「Word for Windows Ver1.2」対応の専用アプリケーションプログラム開発支援グループ「WordStation」を発足した。

```
    GET #2, index, recdata
END SUB

'
' キーを入力する
'
FUNCTION GetKeyCode% (str AS STRING)
    DIM i AS INTEGER

    DO
        str = RTRIM$(LTRIM$(GetStr(NX + NW, NY, WHITE, "")))
    LOOP WHILE str = ""
    IF str = QUITCODE THEN
        GetKeyCode% = FALSE
    ELSE
        GetKeyCode% = TRUE
    END IF
END FUNCTION

'
' 指定座標にデフォルト文字列を表示し文字列を入力する
'
FUNCTION GetStr$ (x AS INTEGER, y AS INTEGER, c AS INTEGER, str AS STRING)
    DIM dum AS STRING

    Display x, y, c, str
    LOCATE y, x
    INPUT "", dum
    IF dum = "" THEN
        dum = str
    END IF
    GetStr$ = dum
END FUNCTION

'
' データを入力する
'
SUB InputData (recdata AS RECSTRUCT)
    recdata.address = GetStr(NX + NW, NY + 2, WHITE, recdata.address)
    recdata.postnum = GetStr(NX + NW, NY + 3, WHITE, recdata.postnum)
    recdata.phonenum = GetStr(NX + NW, NY + 4, WHITE, recdata.phonenum)
    recdata.age = VAL(GetStr(NX + NW, NY + 5, WHITE, MID$(STR$(recdata.age), 2)))
    recdata.birthday = GetStr(NX + NW, NY + 6, WHITE, recdata.birthday)
    recdata.blood = GetStr(NX + NW, NY + 7, WHITE, recdata.blood)
    recdata.etc = GetStr(NX + NW, NY + 8, WHITE, recdata.etc)
END SUB

'
' メニューを表示し対応するプロシージャを呼び出す
'
SUB Menu
    DO
        DispMenu
        SELECT CASE VAL(GetChar$)
        CASE RESIST
            ResistProc
        CASE MODIFY
            ModifyProc
        CASE DELETE
            DeleteProc
        CASE SEARCH
            SearchProc
        CASE RECOVER
            RecoverProc
        CASE PDELETE
            PdeleteProc
        CASE QUIT
            QuitProc
            EXIT DO
        END SELECT
    LOOP
END SUB

'
' 修正プロシージャ
'
SUB ModifyProc
    DIM recdata AS RECSTRUCT                    'レコード変数
    DIM inkeycode AS STRING * 20
    DIM index AS INTEGER
    DIM ret AS INTEGER

    OPEN DATAFILE FOR RANDOM AS #2 LEN = 162
    DispScreen MODIFY
    DO UNTIL GetKeyCode%(inkeycode) = FALSE
        EraseBuffer recdata
        IF SearchKey(inkeycode, index, YES) = TRUE THEN
            DO
                EraseMessages
                GetData index, recdata
                DispData recdata
                InputData recdata
                ret = YNC%("変更しますか")
            LOOP WHILE ret = NO
            IF ret = YES THEN
                recdata.customer = inkeycode
                PutData index, recdata
            END IF
        ELSE
            DispError "データがありません。"
            END IF
            EraseMessages
            EraseData
    LOOP
    CLOSE #2
END SUB

'
' 物理削除
'
SUB PdeleteProc
    DIM recdata AS RECSTRUCT                    'レコード変数
    DIM index AS INTEGER
    DIM tmpmax AS INTEGER

    DispScreen PDELETE
    IF YNC%("物理削除を開始しますか") = YES THEN
        EraseMessages
        tmpmax = 0
        OPEN DATAFILE FOR RANDOM AS #2 LEN = 162
        OPEN TMPFILE FOR RANDOM AS #3 LEN = 162
        FOR index = 1 TO maxindex
            IF keytab(index).active = YES THEN
                tmpmax = tmpmax + 1
                keytab(tmpmax) = keytab(index)
                GetData index, recdata
                DispData recdata
                PUT #3, tmpmax, recdata
                EraseData
            END IF
        NEXT index
        maxindex = tmpmax
        WriteIndex
        CLOSE #2
        CLOSE #3
        KILL DATAFILE
        NAME TMPFILE AS DATAFILE
        DispError "データの物理削除作業を終了しました"
    END IF
END SUB

'
' データを指定レコードに書き出す
'
SUB PutData (index AS INTEGER, recdata AS RECSTRUCT)
    PUT #2, index, recdata
END SUB

'
' 終了処理
'
SUB QuitProc
    Display YNCX, YNCY, YELLOW, TITLE + "を終了します"
END SUB

'
' インデックスファイルの読み込み
'
FUNCTION ReadIndex%
    DIM i AS INTEGER

    i = 0
    OPEN INDEXFILE FOR APPEND AS #1
    CLOSE #1
    OPEN INDEXFILE FOR INPUT AS #1
    DO WHILE NOT EOF(1) OR i >= MAXREC - 1
        i = i + 1
        INPUT #1, keytab(i).customer
        INPUT #1, keytab(i).active
    LOOP
    CLOSE #1
    ReadIndex% = i
END FUNCTION

'
' データ復活プロシージャ
'
SUB RecoverProc
    DIM recdata AS RECSTRUCT                    'レコード変数
    DIM inkeycode AS STRING * 20
    DIM index AS INTEGER
    DIM ret AS INTEGER
    DIM touch AS INTEGER

    touch = NO
    OPEN DATAFILE FOR RANDOM AS #2 LEN = 162
    DispScreen RECOVER
    DO UNTIL GetKeyCode%(inkeycode) = FALSE
        EraseBuffer recdata
        IF SearchKey(inkeycode, index, NO) = TRUE THEN
            EraseMessages
            GetData index, recdata
            DispData recdata
            ret = YNC%("データを復活しますか")
            IF ret = YES THEN
                keytab(index).active = YES
                touch = YES
            END IF
```

```
        ELSE
            DispError "データがありません。"
        END IF
        EraseMessages
        EraseData
    LOOP
    IF touch = YES THEN
        WriteIndex
    END IF
    CLOSE #2
END SUB

'
' 登録プロシージャ
'
SUB ResistProc
    DIM recdata AS RECSTRUCT              'レコード変数
    DIM inkeycode AS STRING * 20
    DIM index AS INTEGER
    DIM ret AS INTEGER
    DIM touch AS INTEGER

    touch = NO
    OPEN DATAFILE FOR RANDOM AS #2 LEN = 162
    DispScreen RESIST
    DO UNTIL GetKeyCode%(inkeycode) = FALSE
        EraseBuffer recdata
        IF SearchKey(inkeycode, index, YES) = FALSE THEN
            DO
                EraseMessages
                InputData recdata
                ret = YNC%("登録しますか")
            LOOP WHILE ret = NO
            IF ret = YES THEN
                maxindex = maxindex + 1
                keytab(maxindex).customer = inkeycode
                keytab(maxindex).active = YES
                recdata.customer = inkeycode
                PutData maxindex, recdata
                touch = YES
            END IF
        ELSE
            GetData index, recdata
            DispData recdata
            DispError "登録済です。"
        END IF
        EraseMessages
        EraseData
    LOOP
    IF touch = YES THEN
        WriteIndex
    END IF
    CLOSE #2
END SUB

'
' 指定したキーに対応するインデックスを探し、あったかどうかを返す
'
FUNCTION SearchKey% (str AS STRING, index AS INTEGER, act AS INTEGER)
    DIM i AS INTEGER

    FOR i = 1 TO maxindex
        IF RTRIM$(LTRIM$(str)) = RTRIM$(LTRIM$(keytab(i).customer)) AND keytab(i).
active = act THEN
            index = i
            SearchKey% = TRUE
            EXIT FUNCTION
        END IF
    NEXT i
    SearchKey% = FALSE
END FUNCTION

'
' 検索プロシージャ
'
SUB SearchProc
    DIM recdata AS RECSTRUCT              'レコード変数
    DIM inkeycode AS STRING * 20
    DIM index AS INTEGER
    DIM ret AS INTEGER

    OPEN DATAFILE FOR RANDOM AS #2 LEN = 162
    DispScreen SEARCH
    DO UNTIL GetKeyCode%(inkeycode) = FALSE
        EraseBuffer recdata
        IF SearchKey(inkeycode, index, YES) = TRUE THEN
            EraseMessages
            GetData index, recdata
            DispData recdata
            DispError ""
        ELSE
            DispError "データがありません。"
        END IF
        EraseData
    LOOP
    CLOSE #2
END SUB

'
' インデックスファイルを書き込み
'
SUB WriteIndex
    DIM i AS INTEGER

    OPEN INDEXFILE FOR OUTPUT AS #1
    FOR i = 1 TO maxindex
        WRITE #1, keytab(i).customer
        WRITE #1, keytab(i).active
    NEXT i
    CLOSE #1
END SUB

'
'「Y」「N」「C」だけを受け付けて対応する値を返す
'
FUNCTION YNC% (str AS STRING)
    DIM dum AS STRING

    Display YNCX, YNCY, WHITE, str + " [Y(es)/N(o)/C(ancel)] ? :"
    DO
        dum = LCASE$(GetChar$)
    LOOP UNTIL dum = "y" OR dum = "n" OR dum = "c"
    SELECT CASE dum
    CASE "y"
        YNC% = YES
    CASE "n"
        YNC% = NO
    CASE "c"
        YNC% = CANCEL
    END SELECT
END FUNCTION
```


ワンラインNEWS
セガ・エンタープライゼスはVR（仮想現実感）を取り入れたゲーム機開発用の高性能CG基板を開発した。この技術を使い夏にも業務用3次元CGゲーム機を商品化する。

98活用アセンブラ技法

RS-232C編 第2回

RS-232C デバイスドライバ

星野 操

今回は，前回作成したRS-232CのBIOSドライバを利用して，DOSで使用可能なデバイスドライバを作成する．前回のasync.asmに拡張を加え，さらに，デバイスドライバ部分であるasync.asmを作成して，キャラクタデバイスドライバを完成させる．

ドライバの機能の検討

前回のドライバでは，DOS標準のRSDRV.SYSに組み込まれている機能をサポートしました．しかし，デバイスドライバを作成する上で，ハードウェア部分とのインターフェースをBIOSのみに限定しようとすると，機能の不足する部分がでてきてしまいます．

たとえば，入力バッファをフラッシュするようなファンクションをサポートする必要がでてきます．この場合，受信バッファの中身がなくなるまでBIOSの読み込みファンクションを実行することで目的を達成できますが，あまりスマートなやり方ではありません．BIOSのワークエリアを参照できるのであれば，リングバッファのデータポインタを直接書き換えてしまうのが一番てっとり早い方法です．しかし，いくらBIOS関係のオブジェクトをデバイスドライバにリンクするといっても，BIOSのワークエリアを直接触るのはあまりに美しくありません．

入力バッファについては，BIOSレベルのアクセスでもなんとかならないこともないのですが，出力バッファとなると，どうにも解決のしようがありません．デバイスドライバには，出力についてもキュー内のデータをフラッシュするというファンクションがあります（DOSのどのファンクションコールによって，このデバイスドライバコールが発生するのかは知らないのですが）．データの送信も割り込み駆動で行い，キューイングする今回のBIOSでは，この出力キューを標準的な方法でクリアすることは不可能です．

BIOS機能の拡張

そこで，BIOS機能の拡張を行わないわけにはいかなくなりました．拡張機能としては，このようなキューのクリア動作のほかに，とくに出力バッファに関するものが必要となります．たとえば，バッファの長さや，送信状態などです．受信機能のファンクションコールをする場合には，まずデータバッファにデータがあることを確認する必要があります．そうしないと，呼び出した先でデータが受信されるまで，待ち続けることになってしまいます．送信の場合も同様に，バッファがフルになってしまうと呼び出し先で止まってしまいます．これを避ける必要がある場合には，送信バッファにまだ空きがあることを確認してから出力しないといけません．

さらに，ある種の状況では，現在データが完全に送出済みであるかどうかをチェックする必要が生じてきます．バッファの中のデータは，送信割り込みによってシリアルチップに送られた時点で消えてなくなります．ですから，バッファの中の最後のキャラクタを送出し始める時点で，バッファの中は空になります．しかし，この最後のキャラクタはこの時点では送出は完了していません．ですから，不用意に回線を切る，などの措置をしてはいけなくなります．

シリアルチップの中のデータが送出完了しているかどうかは，チップのステータスで確認することができ，これを取得するファンクションはBIOSにも用意されています．送出割り込みが起動されているのであれば，このステータスビットは常時送信シフトレジスタが空ではない，ということを示しています．しかし，単にこのステータスを見るだけでは不十分です．フローコントロールなどによって送出が一時的に中断している場合，シフトレジスタは空ですが，送信バッファにはデータが残っている可能性があります．つまり，チップのステータスと送信バッファとを両方確認しないといけないのです．これらは，やはり1つの拡張ファンクションとして用意しておくほうが無難です．

ブレーク信号の処理

もうひとつ考えなければいけないのは，ブレーク信号の処理です．ブレークの送出においては，いくつかの方法が考えられます．MNPやV42bisなどのパケットを利用した圧縮機能付きのプロトコルだと，モデム内部に送受信のバッファが用意されています．このバッファ内のデータと，ブレーク信号との関係をどのようにするかというのは，いろいろ規定できるようになっています．

ひとつは，ブレーク信号によって現在のバッファの内容を破壊（フラッシュ）するかどうかであり，もうひとつはブレーク信号をすぐに（送出バッファ内の待ちキャラクタよりも優先して）送るか，イン・シーケンスで送るかどうかです．

今回のデバイスドライバも，同様にブレーク信号の処理をどうするか考えなくてはいけません．ブレーク信号はデータ送出とは非同期にコントロールされると考えなければならないので，データを破壊したくない場合には，キャラクタ送出と同期を取ってブレークコントロールを実行しなければいけません．イン・シーケンスのブレーク送出の場合には，前記の拡張ファンクシ

ワンラインNEWS
沖電気は米ヒューレット・パッカードと情報通信分野で今後相互に製品の購入・販売やライセンス供与などを実施する基本契約を結んだ．

ョンでバッファ内のデータの送出が完全に終了するまで待ち，それからブレーク制御を行うことで目的は達成できます．即時破壊的にブレークを送信する場合には，現在送出中のキャラクタは保存されなくてもよいわけですから，ブレークを送出した後送出バッファをフラッシュすれば問題はありません．

一番難しいのは，即時非破壊的にブレークを送る場合です．これを可能にするためには，現在送出中のキャラクタと同期を取って，送出完了直後にブレークを送り出さないといけません．このために，キャラクタ同期を取るシステムを導入する必要がでてきます．今回は，フローコントロール機能を流用することにしました．

前回のリストで，tx_flowed_offはソフトウェアフローが実行されている時，xoffが要求され，次のキャラクタ送出の停止が要求されていることを示すものです．このフラグを立てておけば，送出完了割り込みが発生した時点でtx_int_activeがクリアされ，それによってキャラクタ同期を取ることが可能です．

しかし前回のままだと，同期を取っている最中にたまたまxonキャラクタが受信された場合に，不具合が生じることになります．そこで，tx_flowed_offを単なる0／非0のフラグではなく，ビットマップされたフラグとして解釈するようにプログラムを変更しました．以前の通りのソフトウェアフローは最下位ビットを，そして同期のための待ち状態のためには次のビットを使用します．送信完了割り込みのサービスルーチンにおいては，いずれかのビットがセットされていると送出を中断するようにします．また，それぞれの復旧処理においても，他方のビットが立っている場合には，シリアルチップに対して送出割り込み開始のトリガーをかけることはしないようにします．

拡張BIOSサービスとしては，この同期を取って送出を中断するファンクションと，送出を再開するファンクションを用意することにします．なお，ブレーク送出中でも送信割り込みの発生は抑制されません．ですから，BIOSサービスでブレークを送出する場合には，どのような送信状況であるかによらず，送信割り込み駆動を中断するようにフラグを設定しなければなりません．ただし，完全に同期を取ってしまうと，ハードウェアフローなどの理由によってデータ送出が抑制されている場合には，いつまで経ってもブレーク信号が送出されない可能性があります．ブレーク信号は，このような状況をクリアするために送ることも考えられるので，ブレーク送出時には単にフラグを立てるだけにします．そして，必要な場合はユーザー側で明示的にこの同期ルーチンを利用することにします．

一方ブレークを中止する場合には，無条件で再開するファンクションを呼び出すようにします．このために，同期を取ってからブレークをかけた場合には，明示的に再開ファンクションを呼び出す必要はありませんが，そのほかの理由により一度送信を中断した場合には，かならず再開するファンクションを呼出さないといけません．そうしないと，二度と送出が行われなくなります．

割り込みマスクポートの設定

今回もうひとつ修正したのは，割り込みマスクポートの設定です．送信割り込みを禁止・許可するために，割り込みマスクポートに定数を出力していたのですが，このポートはそのほかにシャットダウンやビープ・プリンタコントロールなどにも使用されています．既出の定数だとビープがなりっぱなしになるという問題があるのですが，これを改定したとしても，ビープ中やプリンタ出力中にシリアル送受信割り込みが発生し，これによってポートの値が変更され不都合が生じるかもしれません．

ですから，単に定数を設定することを避け，一度ポートの内容を読み込んでから，マスクをして適当な割り込みマスクビットを設定した後に書き戻すようにしてあります．

async.asmの変更

async. asmに対する変更点を差分リスト（リスト2）に，最後尾に追加した部分を別のリスト（リスト3）に，変更後のasync. amsをリスト1に示します．差分リストの変更点は，前記のようにフロー制御変数を利用した同期送出停止のための操作と，割り込みマスクポートへの出力部分の変更です．このほかに，オリジナルの先頭部分にあったcondx. incのインクルード命令は取り除いてあります．async. asmはdevstd. asmによってインクルードされたxasync. asm（今回作成するデバイスドライバソースファイル）によって，さらにインクルードされる形態を取りますから，devstdによってインクルードされるcondx. incをここで再び読み込む必要はないためです．

追加変更部分はブレークコントロール以降の後で，そこからファイル終端までを置き換える形で表示してあります．ブレークコントロールの部分は，

リスト1 async.asm

```

;--------------------------------
; tx/rx buffer definition
;

; buffer size contstans
RX_BUF_SIZE     equ     256
TX_BUF_SIZE     equ     256

; actual buffer
rx_buffer       db      RX_BUF_SIZE dup (?)
tx_buffer       db      TX_BUF_SIZE dup (?)

; pointer of buffer
rx_top          dw      rx_buffer
rx_tail         dw      rx_buffer
tx_top          dw      tx_buffer
tx_tail         dw      tx_buffer
tx_int_active   dw      0


;--------------------------------
; flow control definition
;

FLOW_NONE       equ     0
FLOW_HARD       equ     1
FLOW_SOFT       equ     2

flow_type       dw      FLOW_NONE
tx_flowed_off   db      0
rx_flowed_off   db      0
flow_char_required      db      0
flow_char_code  db      0

XON_CODE        equ     11h
XOFF_CODE       equ     13h
RSON_COMMAND    equ     37h
RSOFF_COMMAND   equ     17h


;--------------------------------
; hardware definition
;

DATA_PORT       equ     30h
COMMAND_PORT    equ     32h
INT_PORT        equ     0
MASK_PORT       equ     2
INT_MASK        equ     35h
TIMER_CONTROL   equ     77h
TIMER_COUNT     equ     75h


;--------------------------------
; interrupt definition
;

INT_NUMBER      equ     4
INT_VECTOR      equ     (8+INT_NUMBER)*4
INT_BIT         equ     10h
TXINT_ACTIVE_MASK       equ     5
TXINT_INACTIVE_MASK     equ     1

;--------------------------------
; interrupt interface
;

int_work        struc
int_num         dw      ?
int_func        dw      ?
int_old         dd      ?
int_work        ends

interrupt_list  label   byte
i0c     int_work        <0ch,dummy_int,0>
i0f     int_work        <0fh,dummy2_int,0>
i19     int_work        <19h,bios_entry,0>
        dw      0

dummy_int:
        push    ax
        mov     al,20h
        out     INT_PORT,al
        pop     ax
dummy2_int:
        iret


;--------------------------------
; output interrupt trigger macro
; input:
;       al: data to output

output_trigger  macro
        ; port output
        out     DATA_PORT,al

        ; setup tx interrupt
        mov     tx_int_active,1

        in      al,INT_MASK
        call    waste_time
        and     al,not 7
        or      al,TXINT_ACTIVE_MASK
        out     INT_MASK,al
        endm


;--------------------------------
; tx advance pointer macro
; argument:
;       reg: pointer

tx_advance      macro   reg
        inc     reg
        cmp     reg,offset tx_buffer+TX_BUF_SIZE
        @if     ae
                mov     reg,offset tx_buffer
        @endif
        endm


;--------------------------------
; rx advance pointer macro
; argument:
;       reg: pointer

rx_advance      macro   reg
        inc     reg
        cmp     reg,offset rx_buffer+RX_BUF_SIZE
        @if     ae
                mov     reg,offset rx_buffer
        @endif
        endm


;--------------------------------
; interrupt support routine
; broken:
;       ax,bx

rxintr  proc    near
        ; data read
        in      al,DATA_PORT

        ; check flow control char
        cmp     flow_type,FLOW_SOFT
        @if     z
                ; check xoff
                cmp     al,XOFF_CODE
                @if     z
                        ; flow off
                        or      tx_flowed_off,1
                        pop     bx
                        ret
                @endif

                ; check xon
                cmp     al,XON_CODE
                @if     z
                        ; flow on
                        cmp     tx_flowed_off,1
                        @if     z
                                ; output char if need
                                cmp     tx_int_active,0
                                @if     z
                                        ; first check flow character
                                        mov     bx,tx_top
                                        cmp     bx,tx_tail
                                        @if     z
                                                ; no data to output, shortcut
                                                ret
                                        @endif

                                        ; take data
                                        mov     al,cs:[bx]

                                        ; advance pointer and save
                                        tx_advance <bx>
                                        mov     tx_top,bx

                                        ; output data
                                        output_trigger
                                @endif
                        @endif

                        and     tx_flowed_off,not 1
                        ret
                @endif
        @endif

        ; store it into buffer
        mov     bx,rx_tail
        mov     cs:[bx],al

        ; advance pointer
        rx_advance <bx>

        cmp     bx,rx_top
        @if     nz
                ; more buffer space
                mov     rx_tail,bx

                ; check flow
                sub     bx,rx_top
                @if     c
                        ; adjust
                        add     bx,RX_BUF_SIZE
                @endif

                @mc
                        cmp     bx,(RX_BUF_SIZE*2)/3
                        @cond   ae
                        cmp     rx_flowed_off,0
                        @cond   z
                        cmp     flow_type,FLOW_NONE
                        @cond   nz
                @if
                        ; flow off control require
```


ワンラインNEWS
富士通はアクセス時間が15nsと高速の4MビットSRAMを開発、発売した。大型コンピュータのメインメモリや高級EWSのキャッシュメモリ用に売り込む。

```
                                mov     rx_flowed_off,1
                                cmp     flow_type,FLOW_HARD
                                @if     z
                                        ; hardware flow, off RS
                                        mov     al,RSOFF_COMMAND
                                        out     COMMAND_PORT,al
                                @else
                                        ; software flow
                                        cmp     tx_int_active,0
                                        @if     nz
                                                ; already interrupt active
                                                mov     flow_char_required,1
                                                mov     flow_char_code,XOFF_CODE
                                        @else
                                                ; directly output
                                                mov     al,XOFF_CODE
                                                output_trigger
                                        @endif
                                @endif
                        @endif
                @endif

                ret
rxintr  endp


;--------------------------------
; tx interrupt support routine
; broken:
;               ax, bx

txintr  proc
                ; first, check flow character
                cmp     flow_char_required,0
                @if     nz
                        ; output flow character
                        mov     al,flow_char_code
                        out     DATA_PORT,al

                        ; and immidiately return
                        ret
                @endif

                ; check flow control off or buffer empty
                @mc
                        cmp     tx_flowed_off,0
                        @cond   z
                        mov     bx,tx_top
                        cmp     bx,tx_tail
                        @cond   nz
                @if
                        ; take first data from buffer, and output
                        mov     al,cs:[bx]
                        out     DATA_PORT,al

                        ; advance pointer
                        tx_advance <bx>
                        mov     tx_top,bx
                @else
                        ; no data active or flowed off, disable interrupt
                        mov     tx_int_active,0
                        in      al,INT_MASK
                        call    waste_time
                        and     al,not 7
                        or      al,TXINT_INACTIVE_MASK
                        out     INT_MASK,al
                @endif

                ; return
                ret
txintr  endp


;--------------------------------
; interrupt main routine
;

hard_int:
                ; for insurance
                cli

                ; save registers
                push    ax
                push    bx

                @begin
                        ; get status flag
                        in      al,COMMAND_PORT

                        ; check rx ready
                        test    al,2
                        @if     nz
                                ; rx service
                                call    rxintr
                                @continue
                        @endif

                        ; check tx ready, and interrupt service flag
                        @mc
                                test    al,1
                                @cond   nz
                                cmp     tx_int_active,0
                                @cond   nz
                        @if
                                ; tx service
                                call    txintr
                                @continue
                        @endif

                        ; no requirement to service
                        @break
                @again

                ; send EOI
                mov     al,20h
                out     INT_PORT,al

                ; return
                pop     bx
                pop     ax
                iret


;============================
; BIOS support routine
;============================

waste_time      proc
                jmp     $+2
                jmp     $+2
                jmp     $+2
                jmp     $+2
                jmp     $+2
                ret
waste_time      endp

;--------------------------------
; initialize routine
; input:
;               60:68 76543210
;                     |||||||x  xflow on(1), off(0)
;                     ||||||X   hard flow on(1), off(0)
;                     ||||xx    7bits(10), 8bits(11)
;                     |||x      parity(1), no parity(0)
;                     ||x       even parity(1), odd parity(0)
;                     xx        1 stop bit(01), 1.5 stop bit(10), 2 stop bits(11)
;--------------------------------
;               60:69 76543210
;                     ooooxxxx  75b(0001), 300b(0011)...,38400(1010) if available
;                               2nd port not supported

bios_init       proc
                push    cx

                ; ds register is 60h
                ; disable async interrupt
                in      al,MASK_PORT
                or      al,INT_BIT
                call    waste_time
                out     MASK_PORT,al

                ; initialize async
                mov     al,0
                out     COMMAND_PORT,al
                call    waste_time
                out     COMMAND_PORT,al
                call    waste_time
                out     COMMAND_PORT,al
                call    waste_time
                out     COMMAND_PORT,al
                call    waste_time
                mov     al,40h
                out     COMMAND_PORT,al
                call    waste_time

                ; get async command data
                mov     al,ds:[68h]
                mov     ah,al
                and     al,0fch
                or      al,2                            ; x16 mode only
                out     COMMAND_PORT,al
                call    waste_time
                mov     al,RSON_COMMAND
                out     COMMAND_PORT,al

                ; check system clock
                mov     bx,3328
                in      al,42h
                test    al,20h
                @if     z
                        mov     bx,4096
                @endif

                ; calc clock divide rate
                mov     cl,ds:[69h]
                and     cl,0fh
                shr     bx,cl

                ; set clock
                mov     al,0b6h
                out     TIMER_CONTROL,al
                call    waste_time
                mov     al,bl
                out     TIMER_COUNT,al
                call    waste_time
                mov     al,bh
                out     TIMER_COUNT,al

                ; interrupt port set
                in      al,INT_MASK
                call    waste_time
                and     al,not 7
                or      al,TXINT_INACTIVE_MASK
                out     INT_MASK,al

                ; flow mode set
                mov     al,ah
                and     al,3                            ; check flow mode
                cmp     al,3
                @if     z
                        mov     al,0                    ; invalid, no flow
                @endif
                mov     ah,0
                mov     flow_type,ax
```

```

	; work area clear
	mov	ax,offset rx_buffer
	mov	rx_top,ax
	mov	rx_tail,ax
	mov	ax,offset tx_buffer
	mov	tx_top,ax
	mov	tx_tail,ax
	mov	ax,0
	mov	tx_int_active,ax
	mov	tx_flowed_off,al
	mov	rx_flowed_off,al
	mov	flow_char_required,al
	mov	flow_char_code,al
	mov	flow_char_required,al

	; interrupt vector set
	push	ds
	mov	ax,0
	mov	ds,ax
	mov	word ptr ds:[INT_VECTOR],offset hard_int
	mov	word ptr ds:[INT_VECTOR+2],cs
	pop	ds

	; enable mask
	in	al,MASK_PORT
	and	al,not INT_BIT
	call	waste_time
	out	MASK_PORT,al
	mov	ah,0

	pop	cx
	ret
bios_init	endp

;------------------------
; set data into tx buffer
; waste_time if tx buffer is full
; input:
;	cl: data

bios_set_data	proc
	; check tx interrupt
	cli
	mov	al,cl

	@mc
		cmp	tx_int_active,0
		@cond	z
		cmp	tx_flowed_off,0
		@cond	z
	@if
		; directly output
		output_trigger

		; shortcut return
		sti
		ret
	@endif
	sti

	; check buffer space
	@begin
		sti

		mov	bx,tx_tail
		tx_advance <bx>
		cmp	bx,tx_top
	@until	nz

	; set data
	cli
	mov	bx,tx_tail
	mov	cs:[bx],al

	; update pointer
	tx_advance <bx>
	mov	tx_tail,bx

	sti
	mov	ah,0
	ret
bios_set_data	endp

;------------------------
; data get sub function
; return:
;	al: data
;	zflag: set if buffer empty

get_data_sub	proc
	; disable interrupt
	cli

	; check buffer
	mov	bx,rx_top
	cmp	bx,rx_tail
	@if	nz
		; get and save data/flag
		mov	al,cs:[bx]
		push	ax
		pushf

		; advance pointer
		rx_advance <bx>
		mov	rx_top,bx

		; check flow
		@mc
			cmp	rx_flowed_off,0
			@cond	nz

			; no need to check flow type
			mov	ax,rx_tail
			sub	ax,bx
			@if	c
				add	ax,RX_BUF_SIZE
			@endif
			cmp	ax,RX_BUF_SIZE/3
			@cond	b
		@if
			; now flow control on
			cmp	flow_type,FLOW_HARD
			@if	z
				; hard flow, set RS on
				mov	al,RSON_COMMAND
				out	COMMAND_PORT,al
			@else
				; software flow
				cmp	tx_int_active,0
				@if	nz
					; set data
					mov	flow_char_required,1
					mov	flow_char_code,XON_CODE
				@else
					; directly output
					mov	tx_int_active,1
					in	al,INT_MASK
					call	waste_time
					and	al,not ?
					or	al,TXINT_ACTIVE_MASK
					out	INT_MASK,al
				@endif
			@endif

			mov	rx_flowed_off,0
		@endif

		; resume flag and data
		popf
		pop	ax
	@endif

	sti
	mov	ah,0
	ret
get_data_sub	endp


bios_get_data	proc
	@begin
		call	get_data_sub
	@until	nz
	ret
bios_get_data	endp


;------------------------
; get status
;

bios_get_status proc
	in	al,COMMAND_PORT
	mov	cl,al
	mov	ah,0
	ret
bios_get_status endp


;------------------------
; get data counts
;

bios_get_count	proc
	mov	cx,rx_tail
	sub	cx,rx_top
	@if	b
		add	cx,RX_BUF_SIZE
	@endif

	mov	ah,0
	ret
bios_get_count	endp


;------------------------
; break control
;

bios_set_break	proc
	or	tx_flowed_off,2

	mov	al,3fh
	out	COMMAND_PORT,al
	ret
bios_set_break	endp


bios_reset_break	proc
	mov	al,37h
	out	COMMAND_PORT,al

	call	exbios_continue_transmittion

	ret
bios_reset_break	endp


;***************************
; extended service
;

```


ワンラインNEWS
日本フェニックスはまとまったデータをコピー感覚で簡単に転送できるシステムを開発した．パソコンと周辺機器を接続する手法を応用した．

前記のように送信割り込みをコントロールする機能が付け加えられています． 追加された拡張BIOSコールは，exbiosで始まる名前でproc定義してあります．最初のinput_senseは，非破壊読み出しをサポートするものです．バッファ中のデータの有無は別のBIOSコールで把握することにして，ここではデータがない場合に特別な処理をすることはしていません．データが存在すれば，先頭データをclレジスタに設定し復帰するだけです．

その後のinput_flush／output_flushは，前記のようにバッファをフラッシュするためのサービスです．これらはDOSのファンクションコールから直接使用されるかどうかは微妙なところですが，BIOSコールなどで直接使用する分には有効なサービスです．次のget_output_countは入力バッファの中のデータ数を把握する標準

```
exbios_input_sense      proc
        ; check input data
        cli
        mov     ax,rx_top
        cmp     ax,rx_tail
        sti

        @if     nz
                ; get 1st data in buffer
                mov     bx,rx_top
                mov     cl,cs:[bx]
                mov     ch,0
                mov     ah,0
        @endif
        ret
exbios_input_sense      endp

exbios_input_flush      proc
        ; reset input pointer
        cli
        mov     rx_top,offset rx_buffer
        mov     rx_tail,offset rx_buffer
        sti
        ret
exbios_input_flush      endp

exbios_output_flush     proc
        ; reset output pointer
        cli
        mov     tx_top,offset tx_buffer
        mov     tx_tail,offset tx_buffer
        sti
        ret
exbios_output_flush     endp

exbios_get_output_count proc
        ; get data count in the output buffer
        mov     cx,tx_tail
        sub     cx,tx_top
        @if     b
                add     cx,TX_BUF_SIZE
        @endif

        mov     ah,0
        ret
exbios_get_output_count endp

exbios_output_status    proc
        ; cl=2 for output busy, cl=0 for buffer empty, 1 for other
        call    exbios_get_output_count

        or      cx,cx
        @if     z
                in      al,COMMAND_PORT
                test    al,4
                @if     z
                        ; now last char transmitting
                        mov     cx,1
                @endif
        @else
                cmp     cx,TX_BUF_SIZE-1
                @if     ae
                        ; buffer full
                        mov     cx,2
                @else
                        mov     cx,1
                @endif
        @endif

        mov     ah,0
        ret
exbios_output_status    endp

exbios_suspend_transmittion     proc
        ; set suspend flag
        or      tx_flowed_off,2

        ; wait till tx interrupt is halt
        @begin
                test    tx_int_active,0
        @until  z

        ret
exbios_suspend_transmittion     endp

exbios_continue_transmittion    proc
        cli
        @mc
                cmp     tx_flowed_off,2
                @cond    z
                cmp     tx_int_active,0
                @cond    z
        @if
                ; now trigger tx interrupt if need
                mov     bx,tx_top
                cmp     bx,tx_tail
                @if     nz
                        ; take data
                        mov     al,cs:[bx]

                        ; advance pointer and save
                        tx_advance <bx>
                        mov     tx_top,bx

                        ; output data
                        output_trigger
                @endif
        @endif

        ; clear flag
        and     tx_flowed_off,not 2

        sti
        ret
exbios_continue_transmittion    endp


;===========================
; service routine
;

entry_array     label   word
        dw      bios_init               ; code 0
        dw      bios_set_data           ; code 1
        dw      bios_get_data           ; code 2
        dw      bios_get_status         ; code 3
        dw      bios_get_count          ; code 4
        dw      bios_set_break          ; code 5
        dw      bios_reset_break        ; code 6
entry_array_end label   word

EXSERVICE_OFFSET        equu    40h

exentry_array   label   word
        dw      exbios_input_sense
        dw      exbios_input_flush
        dw      exbios_output_flush
        dw      exbios_get_output_count
        dw      exbios_output_status
        dw      exbios_suspend_transmittion
        dw      exbios_continue_transmittion
exentry_array_end       label   word

bios_entry:
        ; first, check ds value
        push    ax
        mov     ax,ds
        cmp     ax,60h
        pop     ax
        @if     nz
                ; call rom bios
                jmp     i19.int_old
        @endif

        ; check command range
        cmp     ah,(offset entry_array_end-offset entry_array)/2
        @if     b
                ; calculate entry address
                push    bx
                mov     bl,ah
                mov     bh,0
                add     bx,bx
                call    entry_array[bx]
                pop     bx
        @else
        sub     ah,EXSERVICE_OFFSET
        cmp     ah,(offset exentry_array_end-offset exentry_array)/2
        @e_if   b
                ; extended service
                push    bx
                mov     bl,ah
                mov     bh,0
                add     bx,bx
                call    exentry_array[bx]
                pop     bx
        @endif

        ; terminate
        iret
```

BIOSのサービスと同様に，出力バッファの中のデータ数を把握するものです．

しかしこれだけでは，出力バッファ長がわからないと，これがフルになっているかどうかなどの判断には使用できません．このため，別にoutput_statusというファンクションを用意しておきます．これはclレジスタに送信状態を返します．これが2であればバッファがフルであることを示し，送信要求はバッファが空くまで保留されます．1であれば送信キャラクタが存在している状態です．ただし，xflowなどによって送信が一時中断されている可能性もあります．これが0であれば，シリアルチップの中にも送出中のデータは存在していないことを示しています．ですから，バッファ内の全データが送出されることに対して同期を取るためには，この値が0になるまで待てばよいことになります．

次のsuspend_transmittionとcontinue_transmittionが，割り込み駆動時のキャラクタ同期を取るためのサービスです．suspendはフローコントロールのフラグを設定した後，割り込み駆動中を示すフラグがクリアされるまで待ち状態に入ります．よって，このファンクションから復帰した時点でチップ内のシフトレジスタはクリアされ，送信割り込みは停止していることが保証されます．

リスト2　async.mod

```
*** f:YRCAA.AAA        Mon Apr 13 06:08:20 1992
--- async.asm      Sun Apr 12 12:43:16 1992
***************
*** 1,5
-           include condx.inc

      ;-----------------------------
      ; tx/rx buffer definition


--- 1,4 -----

      ;-----------------------------
      ; tx/rx buffer definition
***************
*** 99,106

            ; setup tx interrupt
            mov     tx_int_active,1
!           mov     al,TXINT_ACTIVE_MASK
            out     INT_MASK,al
            endm


--- 98,109 -----

            ; setup tx interrupt
            mov     tx_int_active,1
!
+           in      al,INT_MASK
+           call    waste_time
+           and     al,not 7
+           or      al,TXINT_ACTIVE_MASK
            out     INT_MASK,al
            endm

***************
*** 148,155
                    cmp     al,XOFF_CODE
                    @if     z
                            ; flow off
!                           mov     tx_flowed_off,1
                            pop     bx
                            ret
                    @endif

--- 151,158 -----
                    cmp     al,XOFF_CODE
                    @if     z
                            ; flow off
!                           or      tx_flowed_off,1
                            pop     bx
                            ret
                    @endif
***************
*** 157,165
                    cmp     al,XON_CODE
                    @if     z
                            ; flow on
!                           cmp     tx_flowed_off,0
-                           @if     nz
                                    ; output char if need
                                    cmp     tx_int_active,0
                                    @if     z

--- 160,168 -----
                    cmp     al,XON_CODE
                    @if     z
                            ; flow on
!                           cmp     tx_flowed_off,1
+                           @if     z
                                    ; output char if need
                                    cmp     tx_int_active,0
                                    @if     z
***************
*** 182,189
                                    @endif
                            @endif

!                           mov     tx_flowed_off,0
                            ret
                    @endif
            @endif

--- 185,192 -----
                                    @endif
                            @endif

!                           and     tx_flowed_off,not 1
                            ret
                    @endif
            @endif
***************
*** 276,283
            @else
                    ; no data active or flowed off, disable interrupt
                    mov     tx_int_active,0
!                   mov     al,TXINT_INACTIVE_MASK
                    out     INT_MASK,al
            @endif


--- 279,289 -----
            @else
                    ; no data active or flowed off, disable interrupt
                    mov     tx_int_active,0
!                   in      al,INT_MASK
+                   call    waste_time
+                   and     al,not 7
+                   or      al,TXINT_INACTIVE_MASK
                    out     INT_MASK,al
            @endif

***************
*** 421,428
            out     TIMER_COUNT,al

            ; interrupt port set
!           mov     al,TXINT_INACTIVE_MASK
            out     INT_MASK,al

            ; flow mode set

--- 427,437 -----
            out     TIMER_COUNT,al

            ; interrupt port set
!           in      al,INT_MASK
+           call    waste_time
+           and     al,not 7
+           or      al,TXINT_INACTIVE_MASK
            out     INT_MASK,al

            ; flow mode set
***************
*** 572,579
                                    @else
                                            ; directly output
                                            mov     tx_int_active,1
!                                           mov     al,TXINT_ACTIVE_MASK
                                            out     INT_MASK,al
                                    @endif
                            @endif

--- 581,591 -----
                                    @else
                                            ; directly output
                                            mov     tx_int_active,1
!                                           in      al,INT_MASK
+                                           call    waste_time
+                                           and     al,not 7
+                                           or      al,TXINT_ACTIVE_MASK
                                            out     INT_MASK,al
                                    @endif
                            @endif
```


ワンラインNEWS
シャープは欧米向けにポケットサイズ・パソコン「PC-3000」の輸出を開始した．同社は海外でノート型パソコンを販売しているが，販売台数が伸び悩んでいる．

通常の状態で最大１キャラクタ分の遅延が生じますが，ハードウェアフローが使用されている場合では，これがクリアされるまで帰ってこない状態になることもあります．このため，即時破壊的なブレークを送出する必要があ

リスト3　async.mo2

```
;-------------------------
; break control
;

bios_set_break  proc
        or      tx_flowed_off,2

        mov     al,3fh
        out     COMMAND_PORT,al
        ret
bios_set_break  endp

bios_reset_break        proc
        mov     al,37h
        out     COMMAND_PORT,al

        call    exbios_continue_transmittion

        ret
bios_reset_break        endp


;****************************
; extended service
;

exbios_input_sense      proc
        ; check input data
        cli
        mov     ax,rx_top
        cmp     ax,rx_tail
        sti

        @if     nz
                ; get 1st data in buffer
                mov     bx,rx_top
                mov     cl,cs:[bx]
                mov     ch,0
                mov     ah,0
        @endif
        ret
exbios_input_sense      endp

exbios_input_flush      proc
        ; reset input pointer
        cli
        mov     rx_top,offset rx_buffer
        mov     rx_tail,offset rx_buffer
        sti
        ret
exbios_input_flush      endp

exbios_output_flush     proc
        ; reset output pointer
        cli
        mov     tx_top,offset tx_buffer
        mov     tx_tail,offset tx_buffer
        sti
        ret
exbios_output_flush     endp

exbios_get_output_count proc
        ; get data count in the output buffer
        mov     cx,tx_tail
        sub     cx,tx_top
        @if     b
                add     cx,TX_BUF_SIZE
        @endif

        mov     ah,0
        ret
exbios_get_output_count endp

exbios_output_status    proc
        ; cl=2 for output busy, cl=0 for buffer empty, 1 for other
        call    exbios_get_output_count

        or      cx,cx
        @if     z
                in      al,COMMAND_PORT
                test    al,4
                @if     z
                        ; now last char transmitting
                        mov     cx,1
                @endif
        @else
                cmp     cx,TX_BUF_SIZE-1
                @if     ae
                        ; buffer full
                        mov     cx,2
                @else
                        mov     cx,1
                @endif
        @endif

        mov     ah,0
        ret
exbios_output_status    endp

exbios_suspend_transmittion     proc
        ; set suspend flag
        or      tx_flowed_off,2

        ; wait till tx interrupt is halt
        @begin
                test    tx_int_active,0
        @until  z

        ret
exbios_suspend_transmittion     endp

exbios_continue_transmittion    proc
        cli
        @mc
                cmp     tx_flowed_off,2
                @cond   z
                cmp     tx_int_active,0
                @cond   z
        @if
                ; now trigger tx interrupt if need
                mov     bx,tx_top
                cmp     bx,tx_tail
                @if     nz
                        ; take data
                        mov     al,cs:[bx]

                        ; advance pointer and save
                        tx_advance <bx>
                        mov     tx_top,bx

                        ; output data
                        output_trigger
                @endif
        @endif

        ; clear flag
        and     tx_flowed_off,not 2

        sti
        ret
exbios_continue_transmittion    endp


;=========================
; service routine
;

entry_array     label   word
        dw      bios_init               ; code 0
        dw      bios_set_data           ; code 1
        dw      bios_get_data           ; code 2
        dw      bios_get_status         ; code 3
        dw      bios_get_count          ; code 4
        dw      bios_set_break          ; code 5
        dw      bios_reset_break        ; code 6
entry_array_end label   word

EXSERVICE_OFFSET        equ     40h

exentry_array   label   word
        dw      exbios_input_sense
        dw      exbios_input_flush
        dw      exbios_output_flush
        dw      exbios_get_output_count
        dw      exbios_output_status
        dw      exbios_suspend_transmittion
        dw      exbios_continue_transmittion
exentry_array_end       label   word

bios_entry:
        ; first, check ds value
        push    ax
        mov     ax,ds
        cmp     ax,60h
        pop     ax
        @if     nz
                ; call rom bios
                jmp     i19.int_old
        @endif

        ; check command range
        cmp     ah,(offset entry_array_end-offset entry_array)/2
        @if     b
                ; calculate entry address
                push    bx
                mov     bl,ah
                mov     bh,0
                add     bx,bx
                call    entry_array[bx]
                pop     bx
        @else
                sub     ah,EXSERVICE_OFFSET
                cmp     ah,(offset exentry_array_end-offset exentry_array)/2
        @e_if   b
                ; extended service
                push    bx
                mov     bl,ah
                mov     bh,0
                add     bx,bx
                call    exentry_array[bx]
                pop     bx
        @endif

        ; terminate
        iret
```

る場合には，このサービスを呼び出さないで直接ブレークをコントロールします．

continueは前記のフラグをクリアし，そしてほかの理由によって割り込みが禁止されていない場合には，送出割り込みを再開する動作をします．このファンクションはブレーク解除によって自動的に実行されますので，ブレークに関わる処理によってsuspendを実行した場合には，このファンクションは明示的に呼び出す必要はありません．

拡張BIOSの処理テーブル

BIOSのエントリ部分がありますが，拡張BIOSのための処理テーブルが追加されています．既存のBIOSのすぐ後に機能を追加すると，既存のBIOSが拡張された場合などに不都合が生じます．ですから，拡張機能は既存のBIOS処理コードとは離れた値を使用するほうが便利です．ここでは，拡張BIOS機能コードは40Hから始まるようにしてあります．

BIOS処理部分では，標準BIOSの機能コードでなかった場合，拡張BIOSのオフセットを補正した後テーブル参照によって拡張BIOSを呼び出します．

不完全割り込み

最後に，前回説明をしなかった不完全割り込みについて，説明をしておきましょう．今回作成した割り込み処理ルーチンでは，割り込みを要求する諸源がなくなるまでループを繰り返し，対応する割り込み処理をするようにしてあります．いろいろな割り込みの発生タイミングを考えると，これが一番確実に処理できるように思われます．

ただし，このような処理をすると本来割り込みでサービスされるべき処理が，新たな割り込みの発生を伴わずに処理される可能性があります．具体的には，受信割り込みが発生したためそれを処理し，データを取得した（割り込みがインアクティブになった）直後に送信割り込みが発生した場合などです．この場合は，見かけ上割り込みが継続して発生していますが，実際には一度割り込みがインアクティブになっているので，割り込みコントローラーは新たな割り込みの予約を受け付けています．この状態で割り込みを終了すると，割り込みコントローラーは新たな割り込みを発生させます．

しかし，この時点ではシリアルの割り込み要求はすべて解除されています．そこで，条件によって不完全割り込みが発生する可能性があります．ここでは，割り込み番号でOFの処理ルーチンが呼び出されます．これに対応するハードウェア割り込み諸源は用意されていない（スレーブ割り込みコントローラーは別に割り込みベクタを持っている）ので，ここでは単にiretすればよいのですが，BIOSのデフォルトルーチンではEOIをコントローラーに送るようになっています．これが条件によっては不具合の種になってしまいます．

そこで安全のために，今回のドライバでは不完全割り込みの処理ルーチンを自前のルーチン（単にiretするだけ）に置き換えておきます．割り込みベクタの処理が3つ行われているのは，ハードウェア割り込み・BIOS割り込みに加えて，この不完全割り込みの処理ルーチンを設定しているためです．

xasync.asm

さて，以上でasync. asmの説明を終り，次にドライバ部分であるxasync. asm（リスト4）に移ります．このドライバも，テープデバイスと同様なキャラクタデバイスですが，テープデバイスよりはきちんと各ルーチンをサポートしています．

テープデバイスの場合，現実問題として入出力はブロック単位で行うため，バッファのフラッシュや入出力ステータスの報告などのサポートはオミットしてしまっていました．それでも，デバイスをオープンして連続的に読み書きする分にはさほど支障無く使用することができました．

しかし，シリアルデバイスの場合にはもう少し細かい処理が必要となります．たとえば，バッファの中のデータの有無などをチェックするサービスは，サポートしていないと使い難いものになってしまいます．そこで，今回のドライバはキャラクタデバイスとして必要なすべてのサービスは（実際に使用されることがあるかどうかにはよらず）サポートしてあります．

まず最初に，デバイスドライバのタイプを示すインフォメーション定数が定義されています．これはテープデバイスとは異なり，open／close／removeのファンクションをサポートしないような設定にしてあります．ioctlはサポートしています．デバイス名をAUXとしておくと，プログラム起動時に自動的にこのデバイスはオープンされます．そして，open／close／removeサポートビットが設定されているのに，ファンクションがエラーを返すようになっていると，外部プログ

ワンラインNEWS
日本化薬は64MビットDRAM用フォトマスクを製造するための感光樹脂を開発，感光樹脂メーカーに出荷する．この技術で線幅0.3ミクロンのパターンを作ることができる．

ラムを起動するたびにint24の致命的エラー処理ルーチンが呼び出されることになってしまいます.

サポートされていないファンクションのエントリは，次にリストされていますが，ブロックデバイス関係のものおよびopen／closeなど，サポートしていないことを明示しているものに限っています．その下にBIOSの機能コードの定義があります．ここでは機能が追加されることも考えて，定義にマクロを使用しています．実際の処理ルーチンのラベルとのコンフリクトを避けるため，ラベル名は頭に'i_'が付けられるようになっています．なお，equ文において＋0という演算がなされていることに注意してください．これを単にequ bios_codeとすると，tasmでは文字マクロと解釈され，すべての機能コードが同一の値になってしまいます．＋0を付けることで，数値マクロとしての解釈を行わせます．

それから，BIOS呼び出しのマクロがあります．ここでも，引数に取ったラベルにi_が付加されるようにしておきます．また，呼び出し時のDSレジスタの値は自動的に60Hになるようにしておきます．その下には，ファンクション呼び出し終了時のエントリがあります．送受信状態を示す必要があるため，テープデバイスとは異なりビジービットの立ったステータスを返す必要がでてきます．

ファンクションの定義

そして，各ファンクションのボディ部分の定義となります．そのほとんどはBIOSを呼び出すだけで実現可能なようになっています．まず，最初にioctl_inです．以前に書いたように，ioctl_inもioctl_outも特定のバッファ領域のアドレスが渡されるというだけで，別にバッファ領域のアクセスに関して，読み込みのみ・書き込みのみという制限があるわけではありません．だから，別にioctl_outでデバイスドライバがパラメータを返しても実際問題として障害が生じるわけではありません．またioctl_inを呼び出す時に，パラメータをあらかじめバッファ領域に設定し，これを利用して返すデータを作成するということも可能です．

しかし，今回はioctl_inは読み込まれるべきデータ全部をバッファ領域に設定することにしました．ここでは，入力バッファ・出力バッファの中のデ

リスト4　xasync.asm

```
DEVINFO equ     4000h

scsi_mes        db      'Async Driver',0dh,0ah,'$'

error_command:
media_check:
build_bpb:
device_open:
device_close:
removable_media:
output_until_busy:
deinstall:
        mov     ax,e_bad_command
        jmp     interrupt_end
;

;============================
; bios function code
;

bios_code       =       0
defbios macro   id
        i_&id   equ     bios_code+0
        bios_code       =       bios_code+1
        endm

        defbios BIOS_INIT
        defbios BIOS_SET_DATA
        defbios BIOS_GET_DATA
        defbios BIOS_GET_STATUS
        defbios BIOS_GET_COUNT
        defbios BIOS_SET_BREAK
        defbios BIOS_RESET_BREAK

bios_code       =       40h
        defbios EXBIOS_INPUT_SENSE
        defbios EXBIOS_INPUT_FLUSH
        defbios EXBIOS_OUTPUT_FLUSH
        defbios EXBIOS_GET_OUTPUT_COUNT
        defbios EXBIOS_OUTPUT_STATUS
        defbios EXBIOS_SUSPEND_TRANSMITTION
        defbios EXBIOS_CONTINUE_TRANSMITTION

;============================
; bios calling macro
;

rs_bios macro   id
        push    ds
        mov     ax,60h
        mov     ds,ax
        mov     ah,i_&id
        int     19h
        pop     ds
        endm

;============================
; device driver main part

done_bit        equ     100h
busy_bit        equ     200h

function_done:
        mov     ax,done_bit
        jmp     interrupt_end

function_busy:
        mov     ax,done_bit or busy_bit
        jmp     interrupt_end


;--------------------------
; ioctl input
;
; parameter: intput buffer length:w, output buffer length:w, output status:w

ioctl_in:
        les     bx,packet
        les     bx,es:[bx].trans
        mov     al,es:[bx]

        rs_bios <BIOS_GET_COUNT>
        mov     es:[bx],cx
        inc     bx
        inc     bx

        rs_bios <EXBIOS_GET_OUTPUT_COUNT>
        mov     es:[bx],cx
        inc     bx
        inc     bx

        rs_bios <EXBIOS_OUTPUT_STATUS>
        mov     es:[bx],cx

        jmp     function_done


;--------------------------
; input
;

read:
        les     bx,packet
        mov     cx,es:[bx].count
        les     di,es:[bx].trans

        @begin
                or      cx,cx
        @while  nz
                push    cx
                rs_bios <BIOS_GET_DATA>
                stosb

                pop     cx
```

ワンラインNEWS
日本電気と三菱電機は，日本電気が開発した低電力型DRAMのセカンドソースで合意した．磁気記憶媒体の代替需要を狙った専用メモリ．

```
                dec     cx
        @repeat
        jmp     function_done


;---------------------------
; non destructive input
;

nd_read:
        ; first check data existense
        rs_bios <BIOS_GET_COUNT>
        or      cx,cx
        @if     z
                ; no data in buffer
                jmp     function_busy
        @endif

        ; sense data
        rs_bios <EXBIOS_INPUT_SENSE>

        ; set it into packet
        les     bx,packet
        mov     es:[bx].media,cl

        jmp     function_done


;----------------------------
; read status
;

read_status:
        rs_bios <BIOS_GET_COUNT>
        or      cx,cx
        @if     z
                ; no data
                jmp     function_busy
        @endif

        jmp     function_done


;----------------------------
; read flush
;

read_flush:
        rs_bios <EXBIOS_INPUT_FLUSH>
        jmp     function_done


;----------------------------
; write function
;

write:
write_verify:
        les     bx,packet
        mov     cx,es:[bx].count
        les     di,es:[bx].trans

        @begin
                or      cx,cx
        @while  nz
                push    cx
                mov     cl,es:[di]
                inc     di

                rs_bios <BIOS_SET_DATA>

                pop     cx
                dec     cx
        @repeat
        jmp     function_done


;----------------------------
; write status
;

write_status:
        rs_bios <EXBIOS_OUTPUT_STATUS>
        cmp     cl,2
        @if     z
                jmp     function_busy
        @endif

        jmp     function_done


;----------------------------
; write flush
;

write_flush:
        rs_bios <EXBIOS_OUTPUT_FLUSH>
        jmp     function_done


;----------------------------
; ioctl function
;
; function parameter
;       i: initialize, bios parameter following
;       f: flush buffer, 0 for output, 1 for input
;       b: send break
;       r: rest break
;       s: suspend transmittion
;       c: continue transmittion

ioctl_out:
        les     bx,packet
        les     bx,es:[bx].trans
        mov     al,es:[bx]
        mov     dx,es:[bx+1]
        or      al,20h

        ; parse
        cmp     al,'i'
        @if     z
                ; initialize, get parameter
                mov     dx,es:[bx+1]
                mov     ax,60h
                mov     es,ax
                mov     es:[68h],dx

                rs_bios <BIOS_INIT>
        @else
        cmp     al,'f'
        @e_if   z
                ; flush buffer
                cmp     dx,0
                @if     z
                        ; output
                        rs_bios <EXBIOS_OUTPUT_FLUSH>
                @else
                        rs_bios <EXBIOS_INPUT_FLUSH>
                @endif
        @else
        cmp     al,'b'
        @e_if   z
                ; send break
                rs_bios <BIOS_SET_BREAK>
        @else
        cmp     al,'r'
        @e_if   z
                ; reset break
                rs_bios <BIOS_RESET_BREAK>
        @else
        cmp     al,'s'
        @e_if   z
                ; suspend transmittion
                rs_bios <EXBIOS_SUSPEND_TRANSMITTION>
        @else
        cmp     al,'c'
        @e_if   z
                ; continue transmittion
                rs_bios <EXBIOS_CONTINUE_TRANSMITTION>
        @endif

        jmp     function_done


;--------------------------------
; bios support routine
;

        assume  ds:nothing
        include async.asm
        assume  ds:code

end_label:

ex_init proc    near
        ; first check options
        push    cs
        pop     es
        mov     bx,offset options
        @begin
                ; get argument value
                push    bx
                push    es
                les     bx,[bx].o_arg
                call    atoh
                pop     es
                pop     bx

                ;................
                ; nothing to parse
                ;................

                add     bx,5
                cmp     bx,offset options_tail
        @until  nc

        ; interrupt vector setup
        mov     si,offset interrupt_list
        @begin
                mov     ax,[si].int_num
                or      ax,ax
        @while  nz
                mov     ah,35h
                int     21h
                mov     word ptr [si].int_old,bx
                mov     word ptr [si].int_old+2,es

                mov     dx,[si].int_func
                mov     ah,25h
                int     21h

                add     si,size int_work
        @repeat

        ; hardware setup
        in      al,MASK_PORT
        or      al,INT_BIT
        call    waste_time
        out     MASK_PORT,al

        ret
ex_init endp

```


ワンラインNEWS
ワコムとインテグレイテッドテクノロジーズは手書き入力パソコン「PenTop modelV」を共同開発した.専用の電子ペンを使いディスプレイに手書きで直接入力できる.

ータ数と，出力状態を３つのワード変数としてバッファ領域に設定することにします．ここではDOS呼び出し時に渡されるバッファ長のデータは使用していませんから，必ず６バイト以上の領域を渡してやる必要があります．

その次に読み込みファンクションの定義があります．ここでは，文字数分だけBIOSのget_dataファンクションを呼び出し，それを読み込みバッファに設定していきます．bios_get_dataはデータがバッファ内にないとそこで止まりますから，注意する必要があります．

nd_readは非破壊的読み出しをするものです．DOSの4406Hなどのファンクションでは，この非破壊読み出しファンクションを使用して，デバイスのbusy／readyをチェックしています．ここでは，まず最初に読み込みバッファの長さをチェックし，バッファが空であればbusy状態で復帰します．そうでなければ，拡張BIOSコールを実行し，先頭データを取得してこれをパケットの中に設定します．

read_statusはnd_readの前半部分とほぼ同様な処理を行っています．この場合は，非破壊的にデータを読み出す必要もありません．read_flushは，やはり拡張BIOSコールで目的を実現します．

その後に，出力ファンクションのサポートがあります．出力ファンクションも読み込みファンクションと同様に文字数分だけバッファからデータを取得し，BIOSの出力ファンクションを呼び出します．ここでも，もちろんバッファがフルになると処理が止まってしまうことに注意します．write_statusは，拡張BIOSコールの，ステータスチェック用のファンクションを利用します．そして，これが2，つまりバッファフルの場合にのみbusy状態とし，そうでなければready状態とします．write_flushもread_flushと同様に拡張BIOSコールによってバッファのフラッシュを実行します．

ioctl_outでは，BIOSを利用する各種の処理をサポートします．ここでは，シリアルの初期化，バッファのフラッシュ，ブレークコントロール，送信割り込みのコントロールをサポートしています．たいていのBIOS特殊機能はサポートしてあるので，BIOSを使用しなくてもほとんどの処理が可能となります．

各種バッファのフラッシュは，ほかにデバイスドライバファンクションとしてサポートされています．しかし，DOSの上から明示的にこれらを呼び出すことは困難です．ですから，ドライバとしてのサポートとは別に，ioctlでこれらを利用することができるようにしておきました．

ioctl_outのバッファの利用のしかたは，テープデバイスの時と同様に機能コードを示す頭１文字と，必要ならその後のワードタイプの引数という形にしておきます．初期化ファンクションは，機能コードはｉで示され，BIOSワークエリアに設定すべきデータはこの引数として設定しておきます．フラッシュの場合は，機能コードは単一のｆとし，引数が０／１で出力バッファ・入力バッファを指定してやることにします．

キャラクタと同期を取ってブレークを送出したい場合には，ioctl_outで，まずsuspend機能を呼び出してから，次にbreak設定機能を呼び出すことになります．そして一定時間の値にbreak解除機能をioctl_outで呼び出すと，溜っていたデータの送信が再開されることになります．BIOS自体に機能を追加することになった場合には，それをioctl_outでもサポートできるようにしておくとよいでしょう．

その後に，async.asmをインクルードしてあります．この時，dsにnothingをassumeしてあります．BIOSの実行中は，DSレジスタの値は60Hになっており，セグメントcodeとは別の部分を指しています．ですからこの指定をしておかないと，各種変数にCSプレフィクスが（自動的には）付加されなくなります．async.asmでは，レジスタインダイレクトでバッファをアクセスする部分で，CSプレフィクスを付加してあります．インクルードが終った時点で，dsは再びcodeをassumeしてやります．

最後にハードウェア初期化部分を記述します．このドライバでは，現時点ではパラメータは必要としないのでパーザーのスケルトンだけ記述して，中身を空にしておきます．その後に，レジデントプログラムres3のベクタ設定ルーチンと同様にベクタ記述構造体の内容にしたがって，割り込みベクタを設定するようにします．そして最後に割り込みコントローラーのマスクビットを立てて終了します．これによって，シリアルの割り込みはBIOSの初期化ファンクションが呼び出されるまで禁止されます．

sysファイルの生成

さて，これらのファイルを利用してxasync.sysを作成するためには，次のようにします．まず，devstdの中でxtape.asmをインクルードしている部分のファイル名をxtape.asmからxasync.asmに変更します．あるいは，xasync.asm自身の名前をxtape.asmに変更します．それから，以下のコ

マンドを実行します.

```
tasm /DCHAR_DEV /DDEV NAME ='AUX' / DSTAND_ALONE devstd,xasync
tlink /t xasync,xasync.sys
```

キャラクタデバイス指定, SCSIドライバの検出抑制の指定をし, デバイス名はAUXとします. デバイス名自体は別の名前でも支障はありません. こうして作られたxasync. objをリンクし, 直接sysファイルを作成します. masm／linkを使用する場合には, 一度exeに直してから, exe2binなどのコマンドによってsysファイルに変更するようにします.

テストプログラム miniterm

xasync. sysの動作を確認するために, 非常に簡単なターミナルminiterm. asm(リスト5)を作成してみます. minitermは, キーボードから入力されたデータをシリアルに出力し, 逆にシリアルから入力されたデータをコンソールに出力するというだけの簡単なものです. _cをタイプすることによって, mintermは終了します.

プログラムは, 最初にioctlでシリアルの初期化を実行します. デバイスauxはプログラムの起動時にDOSによって自動的にオープンされ, 3番ハンドルでアクセスできるようになっています. シリアルを簡単に使用する分には, このハンドルを使用できますし, DOSの下位ファンクションで1バイト単位での読み書きを実行することもできます. ただし, coockedモードでオープンされていると思われるので, 必要に応じてrawモードに切り替えてやる必要がでてきます.

初期化モードは, 9600bpsの1ストップビット, パリティなし, ハードウェアフローコントロール付きということにします. 設定パラメーターは, ioctlの引数にセットしておき, 3番ハンドルを利用してioctl呼び出しを実行します. それから, 処理ループに入ります.

まず, AH=6／DL=ffhのダイレクト入力によりキー入力をチェックし, キー入力があれば_cであるかどうかを調べます. そうでなければ, DOSの下位ファンクションでAUXに1文字出力します. 次にioctlでAUXの入力バッファの状態を調べます. この時にも, 同様に3番ハンドルをそのまま使用しています. このioctlファンクションだと, 文字が存在していればalに0ffhが, そうでなければ0が返ります. ですからこれを調べて, 非0の時にはやはりDOSの下位ファンクションで1文字AUXから取り込み, それをダイレクト出力で画面に表示します.

モデムなどを接続してこのプログラムを起動することで, 非常に簡単な端末にすることができます. もちろん, もう少し使用できるものにするためには, ファイルの入出力や各種プロトコルのサポート, そして高速通信に対応するためにDOSのオーバーヘッドを極力避けるための工夫, たとえばioctlを利用した複数文字の直接入出力や, int29hを利用した画面文字出力などをしないといけなくなるでしょう.

機能拡張の余地

シリアルドライバの機能拡張の余地としては, RS-232Cの制御線の状態取得・設定などがあげられるでしょう. とくに98の場合, 信号線は複数のポートに散っていますから, これらを一括して取得できるサービスがあると便利です. DTRなどの設定を可能にした場合, シリアルチップに出力するコマンドを定数でなく変数などにする必要もでてくるでしょう. 現状では, DTR／RTSはつねにON状態に設定されてしまいます.

これらのサポートは, やはり拡張BIOSで行い, それをサポートするファンクションをioctlにインプリメントすることによって, DOSレベル・BIOSレベルいずれでもアクセスできるようにするとよいと思われます.

リスト5 miniterm.asm

```
        dosseg
        .model  small,pascal

        include condx.inc
        include utils.inc

        .stack  1000h
        .code

entry:
        @setup
        push    _argc
        push    _argv
        call    _main
exit:
        dos     <4ch,,,>

        .data
ioctl_work      db      10 dup (?)

        .code
func    main <argc,argv>

        ; initialize rs232c, 9600bps 8n1, hard flow
        mov     ioctl_work,'I'
        mov     ioctl_work+1,4ch
        mov     ioctl_work+2,8

        mov     al,3
        dos     <44h,3,3,offset ioctl_work>

        ; repeat
        @begin
                ; keyboard input check
                dos     <6,,,0ffh>
                @if     nz
                        ; check ^c to terminate
                        cmp     al,3
                        @if     z
                                @break
                        @endif

                        ; send it to aux
                        dos     <4,,,ax>
                @endif

                ; check aux status
                mov     al,6
                dos     <44h,3,,>
                or      al,al
                @if     nz
                        ; input
                        dos     <3,,,>

                        ; and echo
                        dos     <6,,,ax>
                @endif
        @again

        mov     al,0
        dos     <4ch,,,>
endf

        end     entry
```


ワンラインNEWS
アップルコンピュータは名古屋商科大学に,「Macintosh PowerBook 100」を1200台納品した. 販売はキヤノン販売で, パソコンは学生に贈呈される.

松尾治己

リソース作成からデバッグまで，すべてをWindows上で実現！

Microsoft QuickC for Windows

米国Microsoft社から発売されているQuickC for Windowsは，ソースの編集，アイコン，ダイアログボックスなどのリソース作成からコンパイル，デバッグまでをWindows上で行うことができる統合開発環境を備えたC言語処理系である．さらに，QuickC for Windowsには，簡単なCASEツールも付属している．この，安価で高機能なWindows開発システムについて紹介する．

注：この記事は米国で販売されているMicrosoft QuickC for Windowsを使って書かれています．日本で販売される製品については134ページをご覧ください．

Microsoft QuickC for Windows (以下QC/Win) には，4冊のマニュアルと5.25インチのフロッピーディスクが5枚入っている（日本で発売されるQC/Win英語版では3.5インチまたは5インチで供給される）．

マニュアル

4冊のマニュアルは，以下のような内容となっている．

Development Environment Guide

インストール前とインストールしてから読むマニュアルである．インストールに関する説明とGraphical Development Environment（QC/Winの統合環境）の基本的な操作方法を解説している．

C for Windows

C言語プログラミングとWindowsプログラミングを解説したものである．従来のCプログラムをWindows対応にするための「QuickWin」でのプログラミングにも少し触れている．

Toolkit

Dialog Editor，Image Editor，QuickCase:W，Resource Compiler，LINK，LIB，DEFファイルフォーマット，RCファイルフォーマットに関して解説したものである．

Windows Programing Reference

WindowsのシステムファンクションであるAPIとWindowsメッセージのリファレンスである．

インストール

1枚目のディスクに入っているsetup.exe（Windowsプログラム）を実行することで，QC/Winをインストールすることが可能である．

「Space Required」は，選択されているOptionが必要とするハードデ

ィスク容量を表している.「Space」に, Setup対象ディスクの空きスペースも表示されるので, 長々とインストールしたあげくに「ディスク容量がたりません」といわれる心配はないだろう.

すべてをインストールしようとすると14MB程度のディスク容量が必要となるが, setup.exeの起動直後はDOS用ライブラリやSmallモデル以外のライブラリをインストールしない設定になっているので6MB程度となっている.

インストールが終了すると, 図1のようなディレクトリ構成ができあがる.

また, Windowsのプログラムマネージャには, 画面2のようなグループが生成される.

Development Environment

Development Environmentは, QC/Winの中心となるWindows対応のQuickC開発環境である.

Windows環境で動作するQuickC Compiler, Linker, Library Manager, Resource Compilerなどを含んでいる. LinkerやLibrary ManagerはDOS版も添付されているが, CompilerはWindows版のみとなっている. これは, OBJファイルのフォーマットが変更されている可能性を示しているので, もし古いバージョンのLinkerやLibrary Managerを使っているのなら置き換えたほうがいいだろう.

テキストエディタ

QC/Winのテキストエディタは, Windows上のエディタとしては標準的な機能をもつものである. このエディタは, Cのソースコードを対象とする場合に限り, リアルタイムに文法解析を実行し, 表示を色分けするという機能をもっている. 色分けは, 通常のステートメント/Keyword/Identifier/Comment/Number/Real/Stringの7種類に対して行われる. デフォルト設定では, KeywordとCommentのみ色設定がされているので, ここまで解析しているということはわからないだろう. ただ, あまり派手な設定をするとやかましいだけでそれほど効果はないようだ (筆者もおもしろがってやってみたが, 結局元に戻した). リアルタイムに解析するので, 入力するそばから色が変化するのがおもしろかった. この機能によって, 簡単な入力ミスの一部が防げると思われる.

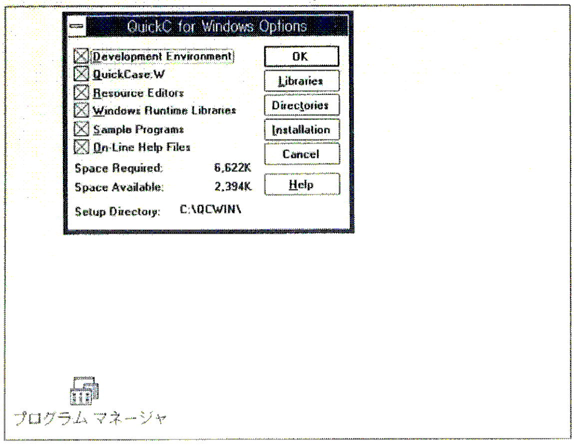
QC/Winには, 最近のWindowsアプリケーションで流行となっているToolbarが設けられている. Toolbar以外には, ステータス表示部分がウィンドウ最下部に設けられている.

Toolbarには, QC/Win内のウィンドウのテキスト表示に使用するフォントの指定と6個のコマンドボタンが用意されている. 漢字を含むフォントを指定すれば, テキストエディタで漢字の表示が可能である. ただし, 文法解析の機能は漢字については考慮していないので, 漢字を使用した部分周辺で正しく色分けされないことがある. ただ, QC/Winのコンパイラ自体は漢字

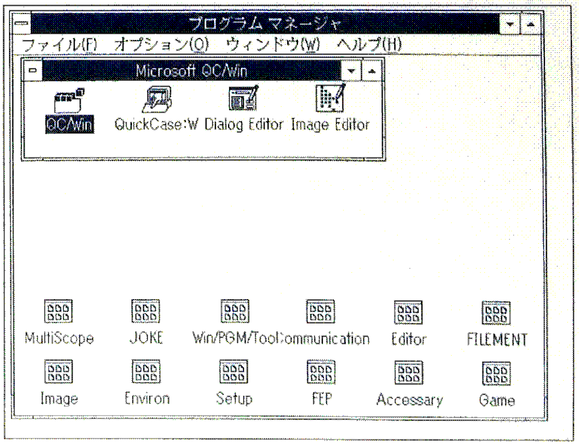
図1　ディレクトリツリー

```
C:¥QCWIN
 ├─BIN
 ├─LIB
 ├─INCLUDE
 │  └─SYS
 └─SOURCE
     ├─BITMAP
     ├─CALC
     ├─CLIPTEXT
     ├─CURSOR
     ├─EDITCNTL
     ├─EDITFILE
     ├─EDITMENU
     ├─FILEOPEN
     ├─GENERIC
     ├─ICON
     ├─INPUT
     ├─MENU
     ├─OUTPUT
     ├─PRNTFILE
     └─SELECT
```

画面1　QuickC for Windows Optionsダイアログ



画面2　Microsoft QC/Win グループ



ワンラインNEWS
ロータスは日本電気のPC-9801 NLに対応した表計算ソフト「1-2-3 R2.3J」のJEIDA規格ICカード版を開発, 販売を開始した.

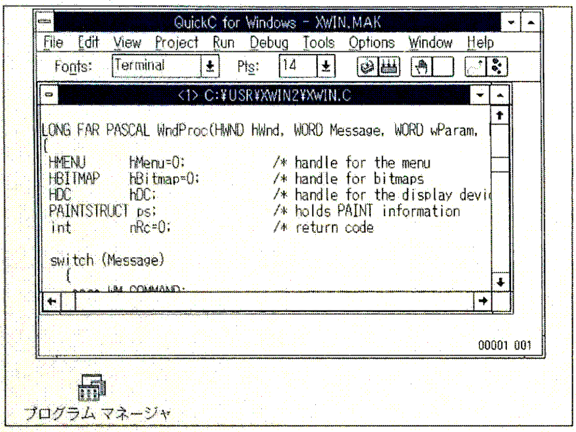
を認識しているので，プログラムの動作上は問題なく漢字の入出力を行うことが可能である．

プロジェクトマネージャ

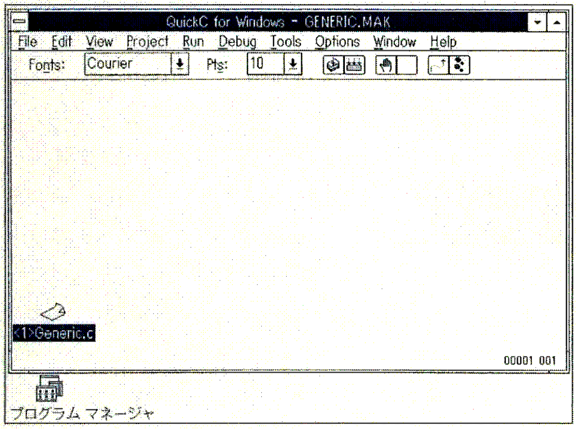
QC/Winでは，プロジェクトマネージャによって，Makefileの作成･管理を行うことが可能である．直接Makefileを記述する必要はない．またQuickCase:WがMakefileのプロトタイプも生成するので，QuickCase:Wで生成したプログラムであれば，これを使用するだけでもよい．プロジェクト管理は，Projectメニューの Editで起動されるダイアログボックスで行うことができる．ただし，ファイル間の依存関係に関しての調整は不可能なので，細かいコンパイル制御をするような用途には向いてないだろう．

最近のMicrosoftの言語製品に添付されている ProgrammersWork

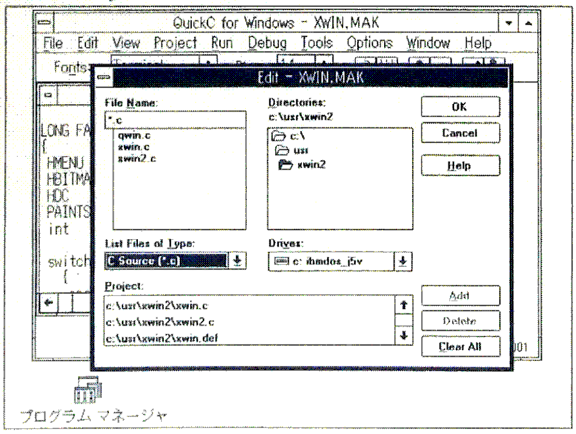
画面3　QC/Winウィンドウ



画面4　QC/WinのToolbar



画面5　Project Editダイアログ



リスト1　Makefile例

```
ORIGIN  = QCWIN
ORIGIN_VER  = 1.00

PROJ    =XWIN
DEBUG   =1
PROGTYPE    =1
CALLER  =
ARGS    =
DLLS    =
CVPACK  =1
CC  =cl -qc
RC  =rc
CFLAGS_G_WEXE   =/AS /W4 /Ze /D_WINDOWS /G2w /Zp
CFLAGS_D_WEXE   =/Zi /Od
CFLAGS_R_WEXE   =/O /Ol /Gs /DNDEBUG
CFLAGS_G_WDLL   =/AS /G2w /Zp /Aw /W3 /D_WINDOWS /D_WINDLL
CFLAGS_D_WDLL   =/Gi /Od /Zi
CFLAGS_R_WDLL   =/O /Os /DNDEBUG
CFLAGS_G_WTTY   =/AS /G2w /W3 /D_WINDOWS
CFLAGS_D_WTTY   =/Gi /Od /Zi
CFLAGS_R_WTTY   =/O /Os /DNDEBUG
CFLAGS_G_DEXE   =/AS /W2
CFLAGS_D_DEXE   =/Gi /Od /Zi
CFLAGS_R_DEXE   =/O /Ot /DNDEBUG
CFLAGS  =$(CFLAGS_G_WEXE) $(CFLAGS_D_WEXE)
LFLAGS_G_WEXE   =/A:16/ST:5120
LFLAGS_D_WEXE   =/CO
LFLAGS_R_WEXE   =
LFLAGS_G_WDLL   =/ST:5120 /A:16
LFLAGS_D_WDLL   =/CO
LFLAGS_R_WDLL   =
LFLAGS_G_WTTY   =/ST:5120 /A:16
LFLAGS_D_WTTY   =/CO
LFLAGS_R_WTTY   =
LFLAGS_G_DEXE   =/NOI /ST:2048
LFLAGS_D_DEXE   =/CO
LFLAGS_R_DEXE   =
LFLAGS  =$(LFLAGS_G_WEXE) $(LFLAGS_D_WEXE)
RCFLAGS =
RESFLAGS    =/t
RUNFLAGS    =
H =     XWIN.H
DEFFILE =   XWIN.DEF
OBJS_EXT =
LIBS_EXT =

.rc.res: ; $(RC) $(RCFLAGS) -r $*.rc

all:    $(PROJ).EXE

XWIN.OBJ:   XWIN.C $(H)

XWIN.RES:   XWIN.RC $(RESFILES) $(H)

XWIN2.OBJ:  XWIN2.C $(H)

$(PROJ).EXE:    XWIN.OBJ XWIN2.OBJ $(OBJS_EXT) $(DEFFILE)
    echo >NUL @<<$(PROJ).CRF
XWIN.OBJ +
XWIN2.OBJ +
$(OBJS_EXT)
$(PROJ).EXE

C:¥QCWIN¥LIB¥+
/NOD slibcew oldnames  libw
$(DEFFILE);
<<
    link $(LFLAGS) @$(PROJ).CRF
    rc $(RESFLAGS) XWIN.RES $(PROJ).EXE

$(PROJ).EXE:    XWIN.RES
    rc $(RESFLAGS) XWIN.RES $(PROJ).EXE

run: $(PROJ).EXE
    $(PROJ) $(RUNFLAGS)

```

Benchのようにソースコード内からinclude指定を検索して,自動的に依存関係を設定してくれればもっといいのだが,そこまではやってくれなかった.

コンパイラ

QC/Winのコンパイラは,DOS版のQuickCコンパイラのアップバージョン（Ver2.60）となっている．コンパイルやリンクのオプション指定は，プログラムの種類を選択した段階で適応したオプションが選択されている．細かいオプションの変更は，Optionsメニューから起動されるダイアログボックスで対話的に行うことが可能である．また，コマンドラインコンパイラで使用する場合のオプション文字列の参照も可能となっている．

QC/Winのコンパイラは,DOS版のQuickCコンパイラ同様にオプティマイズ機能は簡単なものである．製品として流通させるアプリケーションであれば，開発中はQC/Winを使用し，最終的な商品をMicrosoftC Compilerでオプティマイズして出荷するのがよいだろう．

コンパイル速度に関しては，筆者がほかの環境を継続して使用したことがあまりない上に，コンパイラの速度を比較するための適切な基準が存在しないと思っているため，ここでは比較しないことにする．ただ，気になるほど遅くは感じなかった．

しかし，Windowsプログラムにおける大きな難所として「windows.hのコンパイル」がある．Borland C++では，この問題に対処するために，pre-compiled headerという機能を用意した.QuickCでは,インクリメンタルコンパイル機能でとりあえず対処しているような状態であり，この点では不満が残る．米国ではMicrosoftCの7.0が発売され，この7.0のスペックの中にはpre-compiled headerも含まれているので，これにQuickCompile機能をプラスしたならば，かなりの速度アップになるものと思われる．

コンパイル中に発生したErrorやWarningは，Errorsウィンドウに表示される．このエラー報告のフォーマットはDOS版のものと同じである．Errorsウィンドウに報告された行をマウスでダブルクリックするか，テキストカーソルをエラー行に置いて[Enter]キーを押せば，ソースプログラムの該当行にテキストカーソルが移動する．これなどは，BorlandのTurboシリーズやMicrosoftのPWBなどでおなじみの統合開発環境そのものである．

Toolbar上にはCompileとBuildの２つのボタンが用意されているので，この２つに関してはメニューを開いて項目を選択する必要がないのは便利だ．

デバッガ

QC/Winのデバッガは,QC/Winの環境上でそのまま実行可能である．プログラムを切り換えたり，起動したりする手間はまったくない．デバッグ対象のソースコードの表示も，プログラムを入力していたテキストエディタのウィンドウがそのまま使用される．環境の変化が少ないという点で大きく評価できるだろう．筆者が，はじめて触れたWindows上で動作するＣのソースコードデバッガでもある（次に見たのが，MultiScopeDebuggerである．これ以外にはまだ見たことがない）．

デバッガ用にはWatch/Local/Registerのウィンドウをオープンすることが可能である．Watch/Localは変数などのデータ内容の表示を行うウィンドウである．RegisterはCPUのレジスタ内容を表示するウィンドウである．

どちらのウィンドウのデータもデバッガに制御が戻った段階で内容が更新される．

簡単に変数内容を表示するにはQuick Watchダイアログを利用する．

Quick Watchダイアログで参照した変数は，Add Watchボタンによってワンタッチでウィンドウに追加することが可能である．

また,Watchウィンドウへの参照データの追加用のボタンがToolbarに配置されているので活用するとよいだろう(眼鏡のボタン)．追加したい参照データのシンボル名文字列を選択状態にして,Watchボタンをクリックすればよいだけだ.

プログラムのステップ実行には注意が必要である．Windowsのプログラムは，（プログラム側にある）WinMain内のメッセージループによって，Windowsから(プログラム側にある）ウィンドウプロシージャに次々とメッセージが渡されることによって処理が進んでいくという構造になっている．ダイアログボックスもメッセージループがソースコード上に現れないが同様の構造で動作している．

CodeViewなどの今までのデバッガでは，実に素直なトレースをするため，このようなプログラムをデバッグする場合には，メッセージループとウィンドウプロシージャの間に入るWindowsの処理が邪魔になってしまい，単純な関数内に入り込んでトレースするという構造のデバッガでは，ウィンドウプロシージャに到達することはできない．そこで，ウィンドウプロシージャ内にBreakPointを設定して

ワンラインNEWS
横河・ヒューレット・パッカードはキヤノン販売と米ヒューレット・パッカード製WSに関するVAR契約を結んだ．ビジネス分野でWS販売を拡大するのが狙い．

プログラムを実行，Breakした段階からStep実行によってプログラムをトレースするという作業が必要になる．メッセージによるBreakなどもあり得るが，結局はBreakPointの設定を必要とするという点で，手間がかかるのは同じである．QC/Winの将来のバージョンでは，このような機能が追加されて欲しいものである（Multi ScopeDebuggerでは，Traceによってメッセージループからウィンドウプロシージャに入り込むという機能を実現しているので，不可能なことではないと思われる）．

QC/WinのデバッガでのBreak Pointの設定は，ソースコード上でBreakさせたい行にテキストカーソルを移動し，Toolbar上のBreak Pointボタン（手のひらのボタン）をクリックするだけである．さらに細かい設定をしたければ，DebugメニューのBreakPointでオープンするダイアログボックスを使用すればよい．

BreakPointには，Breakする条件設定を細かく指定することが可能である．また，Windows対応のデバッガということで，ウィンドウプロシージャにBreakPointを設定することも可能である．

テキスト表示と併用するようなデバッガとは異なり，Windows上でターゲットプログラムと並行して動作しているので，デバッグがたいへんやりやすかった．また，デバッグ中にほかのWindowsアプリケーションへの切り換えを行っても何の問題も起こらない点も安心である．

QuickCase:W

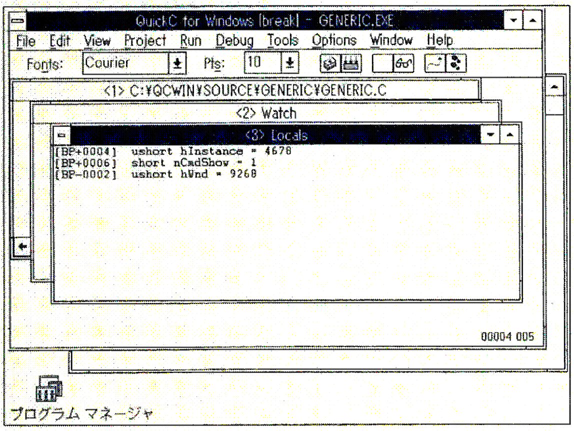
QuickCase:Wは，CASE-WORKS, Inc.が開発したWindows用のCaseツールである．

Caseツールとしてはごく簡単なものだが，Windowsプログラムのユーザーインターフェース部分のプロトタイプを作成してくれるありがたいツールである．

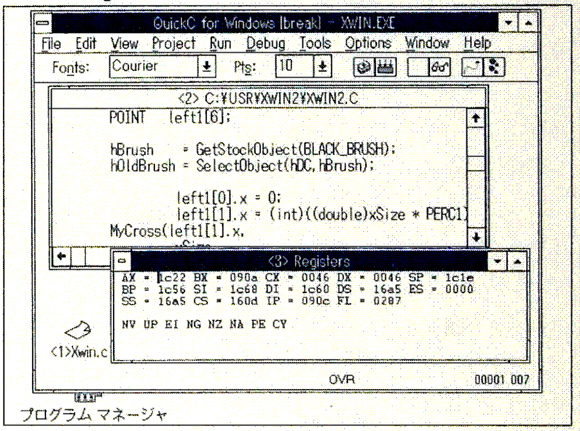
QuickCase:Wには，より多くの機能をもつ上位のバージョンも用意されていて，製品にはパンフレットが同梱されていた．

QuickCase:W上では，表1の設定

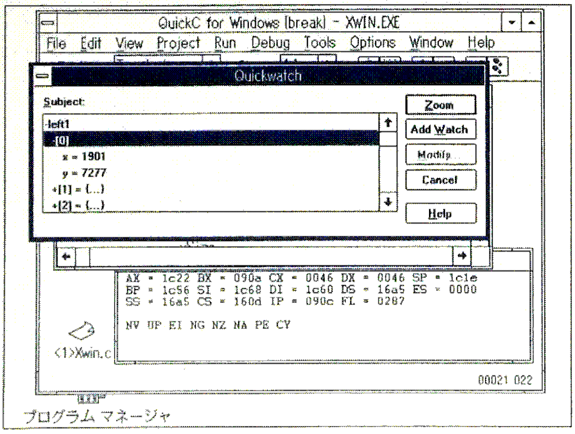
画面6　Watch/Localウィンドウ



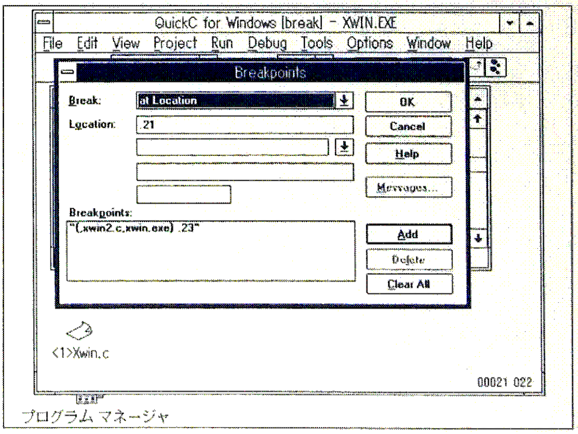
画面7　Registerウィンドウ



画面8　Quick Watchダイアログ



画面9　BreakPointダイアログ



を行うことが可能となっている．そして，その設定を実現するプログラムを生成してくれる．

最後のメニューの設定以外はDesignメニュー内の各項目を選択することで，対応したダイアログボックスウィンドウがオープンし，ダイアログボックス上で設定を行う．最後のメニュー設定は，QuickCase:Wのウィンドウ内に表示されているウィンドウ上のメニューをクリックし，オープンされるダイアログボックスにメニューの設定項目を指定することで設定することになる．

メニュー項目には，StringかBitmapを表示することが可能である．また，メニュー項目間の区切りとしてのSeparatorを指定することも可能である．

また，メニューを選択された場合の挙動を４つの中から選択することができる．

・None

関係付けなし．

・Dropdown Menu

プルダウンメニューと関係づけて，メニューを選択された時にプルダウンメニューがオープンされ，さらに選択を続けられるもの．

・Dialog Box

ダイアログボックスと関係づけて，メニューを選択された時にダイアログボックスがオープンされるもの．QuickCase:Wは，関係づけられたダイアログボックスに対応するダイアログプロシージャのプロトタイプも生成する．

・User-Defined Code

プログラムソースと関係づけて，メニューを選択された時に実行するプログラムを指定するもの．プログラムソースは拡張子.INCのものを指定するようになっていて，リスト２のような形式でプログラム中にインクルードされる．

これらの設定を行ってからBuildメニューのGenerateを選択すると，設定に従ったCのソースプログラムなどが生成される．プログラム生成中は，QuickCase:Wウィンドウ内にGenerateアイコンが表示されている．

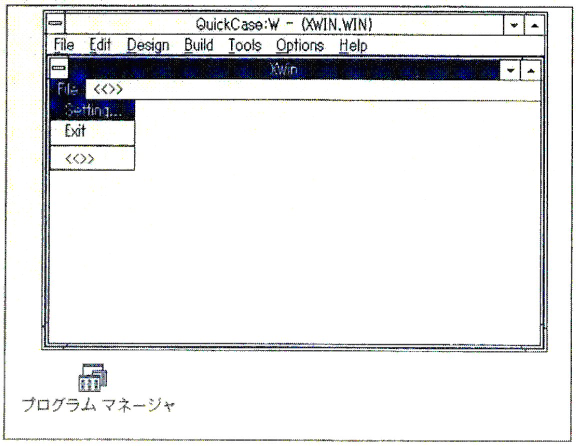
Generateが終了すれば，Windowsプログラミングで必要となる「おまじない」的な部分の記述ができあがったことになる．このソースをプロトタイプとして，プログラム固有の処理を肉付けしていくのである．また，このCのソースプログラムには，Generate時にQuickCase:Wが付けた詳細なコメントも含まれているので,「おまじない」が実際にどのような処理をしている部分なのかを理解する手助けにもなる．コメントは，その詳細度によってLow/Moderate/Highのレベルから選択できるようになっている．

Generateは，最初に１回だけ実行すればよく，その後のQuickCase:Wの変更をソースコードに反映するには，BuildメニューのUpdateを実行する．再度Generateを実行してしまうとソースコードを新規に生成し，上書きしてしまうので，Generate後に変

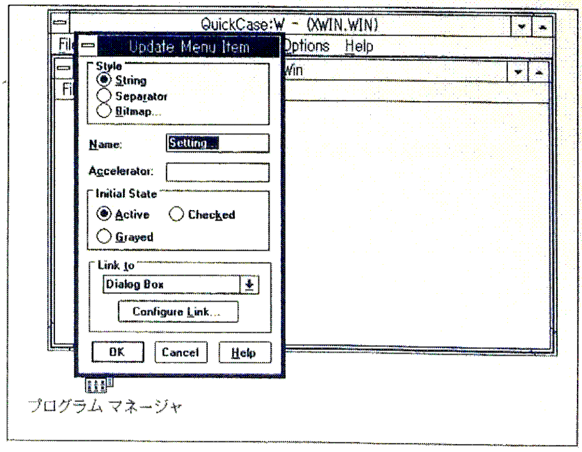
表１

Style	メインウィンドウのスタイル設定
Title	メインウィンドウのタイトル
Icon	アイコン
Cursor	カーソル
Color	前景と背景の色
Scroll Bars	スクロールバーの有無と設定
Size and Location	起動時のウィンドウサイズと位置指定
Unlinked Dialogs	使用するダイアログボックスの指定
該当項目なし	メニューとそこから起動されるダイアログボックスの設定

画面10　QuickCase:Wのウィンドウ



画面11　メニュー設定ダイアログボックス



ワンラインNEWS
日立製作所と米ヒューレット・パッカード（HP）はHPのWSに日立のオブジェクト指向エキスパートシステム構築ツールを英HPを通じて販売する．

更した部分がまったくなくなってしまうから注意が必要である．

QuickCase:Wは，Cソースコード以外に，拡張子.RCのリソースコンパイルファイル，拡張子.DEFのモジュール定義ファイル，拡張子.Hのヘッダファイル（Cソースで使用するID定義や関数プロトタイプ定義），拡張子.MAKのプロジェクトファイル(Makefileといったほうがわかりやすいかな？）を生成してくれる．生成したファイルをそろえてQC/Winでコンパイルすれば，中身はないけれどもとりあえず動作するプログラムが完成する．

Resource Editors

Windowsプログラムで使用するリソースデータを作成，編集するためのツール群である．Bitmap, Icon, Cursorの編集を行うImage Editor, Dialog Boxの編集を行うDialog Editorが含まれている．どちらのEditorも，Microsoft SDKに含まれる同様のエディタのアップグレード版となっている．とくにダイアログエディタのユーザーインターフェースは，大きく刷新されていて，使い勝手が向上している．

Image Editorについては，SDKpaintと機能的にほとんど変わっていない．唯一気がついた変化は，Optionsメニューが新設され，Gridの機能が付加されたということくらいである．

Dialog Editorのほうはかなり変化しているので紹介しておこう．

1つめは，ダイアログボックス内に配置する各コントロールは，メインウィンドウとは別にオープンされるToolboxウィンドウで選択するという点である．

2つめは，コントロールを整列させるための編集機能が用意されたという点である．これらの編集機能はArrangeメニュー内に置かれ，コントロールの配置やコントロール間のスペーシングを整える機能やコントロールを同じ大きさにそろえる機能やPush Buttonコントロールの整列配置を行う機能がある．

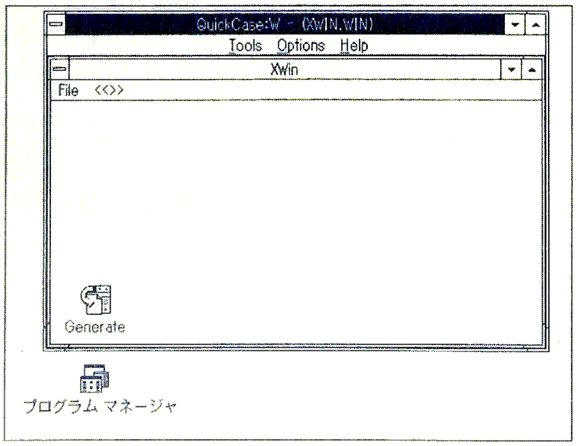
また，注意すべき点もある．それは，1つのプログラムでDialogを複数作成し，使用する場合である．Dialog Editor自身は，1つの定義ファイルに複数のダイアログボックス定義を格納することが可能となっている．しかし，このダイアログボックス定義を，QuickCase:Wから利用しようとすると，QuickCase:Wが.DLG内の複数のダイアログボックス定義を認識できないのである．したがって，QuickCase:Wから利用するためには，1つの.DLG内には1つのダイアログ定義としておく必要がある．QuickCase:Wを使用しないというのであれば，こんなことに注意する必要はない．

また，シンボル定義を保存する拡張子.Hのファイルのファイル名には注

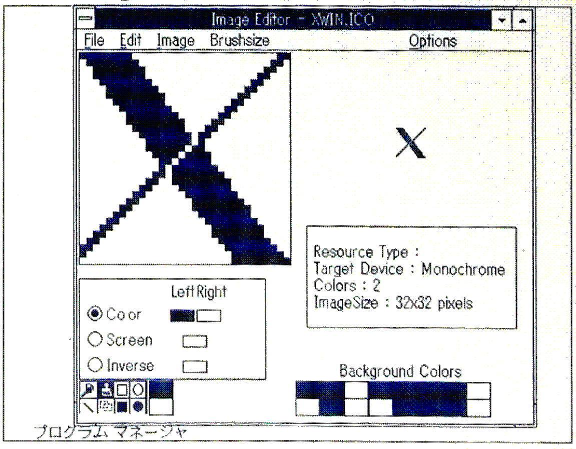
リスト2　プログラムのインクルード部分の抜き出し

```
------------------ include 部分抜粋 ----------------------------------
    case WM_COMMAND:
         switch (wParam)
           {
            case IDM_F_EXIT:
                 /* Place User Code to respond to the              */
                 /* Menu Item Named "Exit" here.                   */
                 #include "XWINEXIT.INC"
                 break;

            default:
                return DefWindowProc(hWnd, Message, wParam, lParam);
           }
         break;        /* End of WM_COMMAND                        */
------------------ xwinexit.inc --------------------------------------
DestroyWindow(hWnd);
if (hWnd == hWndMain)
        PostQuitMessage(0);
----------------------------------------------------------------------
```

画面12　Generateアイコン表示



画面13　Image Editor



意しておいたほうがよい．Quick Case:Wでも拡張子.Hのファイルを作成するため，上書きされてしまう可能性があるからである．

Windows Runtime Library

Windows用のランタイムライブラリである．デフォルトでは，Windows関連のライブラリのSmallモデルのみが選択されている．インストールするLibraryは，Librariesボタンをクリックすることで選択することが可能となる．

ライブラリの種類を見ると，作成可能なプログラムが，DOS版EXE・Windows版EXE・Windows版DLL・QuickWin版EXEの4種類であることがわかる．

DOS版EXEを開発する場合には，Windows上で使用可能な統合開発環境が得られるということになる．ただし，Windows上でデバッグはできない．ただ，CodeView対応のEXEファイルを生成することが可能なので，別製品のデバッガ（CodeViewやTurboDebuggerなど）でデバッグすることは可能である．

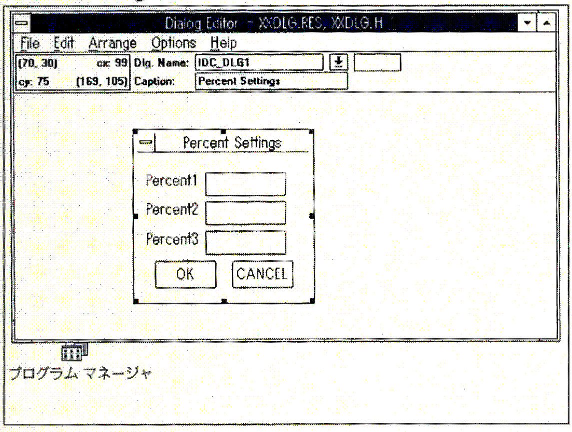
Windows版EXEは，QuickCase:Wのおかげで，複雑なプログラムでなければ意外と簡単に開発することが可能である．

Windows版DLLは，QuickCase:Wがプロトタイプを作ってくれるようなものではないので，Windowsプログラミングに習熟していなければ，作成はむずかしいであろう．DLLのサンプルプログラムも用意されているので，QC/Winを使用してDLLプログラミングを習得するのもいいかもしれない．

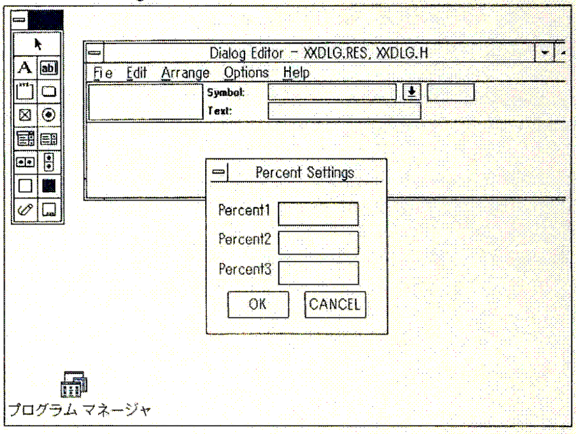
QuickWin版EXEは，printf, scanfを使用したプログラムをWindows上で動作させるための環境で，Windows上にこれらの関数の入出力先が用意されることになる．

QuickWin版EXEを実行すると，Windows上にQuickWinのメインウィンドウがオープンされる．メインウィンドウ上には，stdin/stdout/stderr入出力が割り当てられたウィンドウがオープンされて，DOSのテキスト画面に表示されていたものがウィンドウ内に表示され，Windows上での実行が可能となる．

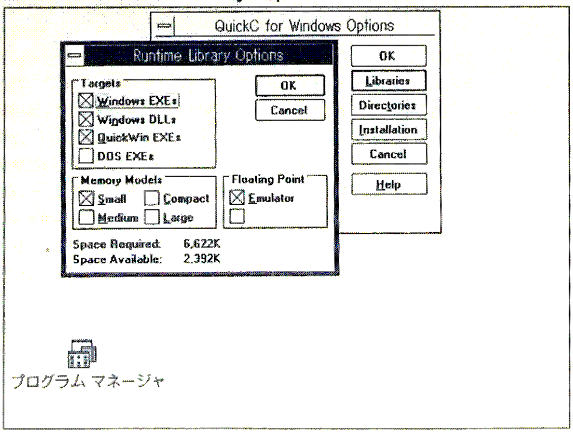
画面14　Dialog Editor



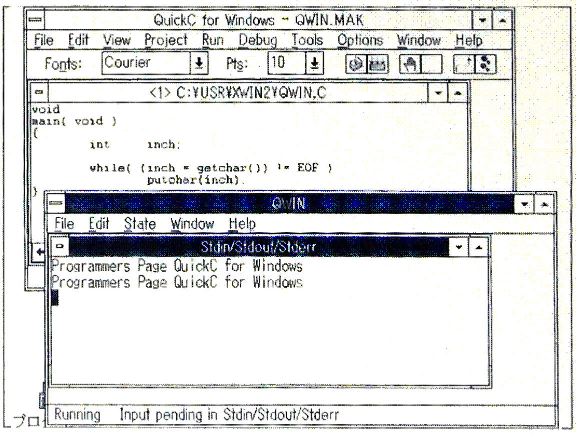
画面15　Dialog EditorのToolboxウィンドウ



画面16　Runtime Library Optionsダイアログ



画面17　QuickWin実行画面



ワンラインNEWS
カシオ計算機はオフコンなど情報機器の国内販売体制を再編する．地域別の情報機器販社を統合した新会社を設立，オフコン向けのソフト開発会社も統合した．

QuickWinプログラムは，標準入出力を使用しているソースをリコンパイルして，ライブラリをQuickWinのものと置き換えるだけで作成することができる．

QuickWinでは，QuickWinウィンドウ内に新たにウィンドウを生成して，Cのストリームとして利用することができるファンクションも用意されているので，「Windowsプログラムはめんどうで組む気にならない」という方は，こちらを利用してみるのもいいかもしれない．

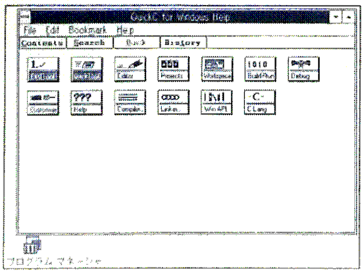
ライブラリにつきもののヘッダファイルに関しては，Microsoft Cの7.0で採用されるはずのC++に関する記述が含まれていたりするのが興味深かった．

Sample Program

QC/Winには，Windowsのサンプルプログラムが15本用意されている．

サンプルには，マニュアルのC for Windowsでの解説に使用しているサンプルプログラムも含まれている．DLLのサンプルプログラムでは，作成するDLLを使用するプログラムとしてVisualBasicのソースが用意されている点などがおもしろかった．まあ，VisualBasicの位置付けをMicrosoftがどのように思っているのかがわかる例といえるだろう．惜しいのは，

画面18　QC/Winのヘルプ画面



QuickCase:Wのサンプルが用意されていないことである．

On-Line Help Files

Windowsのヘルプ形式で作成されたオンラインヘルプで，以下の4本のヘルプが提供されている．

- **DLGEDIT.HLP**
  Dialog Editorのヘルプ
- **QCASEW.HLP**
  QuickCase:Wのヘルプ
- **QCWIN.HLP**
  QC/Winのヘルプ
- **QWIN.HLP**
  QuickWinのヘルプ

QWIN.HLP以外の3本は，日本語Windowsに添付されているWinHelp.exeではバージョンが異なるため参照することができない．QC/Winで使用されているバージョンのヘルプファイルでは，ヘルプ画面にボタンを並べて，選択することができるようになっている．

したがって，このヘルプを参照するためには，QC/Winに添付されているWinHelp.exeを使用しなければならない．QC/Winに添付されているWinHelp.exeは，WindowsのWinHelp.exeと同じ名前でかつWindowsのディレクトリにインストールするようになっているので，インストールする場合には，WindowsのWinHelp.exeに上書きされないように名前を変更するなどして保存しておく必要がある．QC/Win用のWinHelp.exeは日本語を正しく表示できないため，日本語Windowsで使用されている日本語のヘルプがまったく読めなくなってしまう．

注：4月下旬にマイクロソフト（株）から発売された「Microsoft QuickC for Windows」（英語版）では，WinHelp.exeが差し替えられており，上記のような不都合はなくなっている．

使ってみて……

QC/Winはお手軽な値段でWindowsプログラムの開発・デバッグが可能となるたいへん嬉しい製品であった．Windowsプログラムの初心者にもとっつきやすい仕掛がされているのでプログラムを組んでもいいかなという気にさせてくれた．ただ，まだまだWindowsの複雑さがはっきりとプログラマに見えているので，これを覆い隠してくれるような環境が用意されれば，さらにWindowsプログラムは作りやすくなるのではないだろうか？

また，MicrosoftC+SDKという組み合わせで開発をしている上級者にも，Quickコンパイルとインクリメンタルコンパイルを Windows上でサポートし，レスポンスのよい開発・デバッグができる環境としてお勧めしたい製品である．ただ，MicrosoftCの新バージョンでは，SDKが不要で，C++が採用され，Classライブラリ（もちろんWindows対応）が用意されているという話である．QC/Winと，この新バージョンのコンパイラとの関係なども気になるところである．

とりあえずは，日本語に完全対応したQC/Winができるだけ安価に発売されて欲しいものだ．

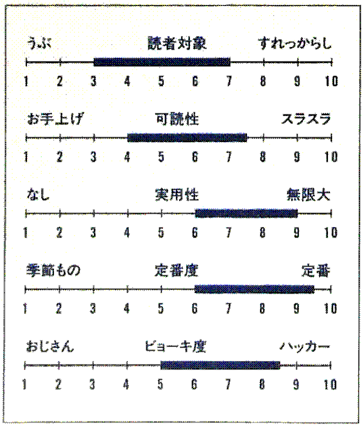
Book Review

■中村正三郎

■アルゴリズム(原書第2版)

R.セジウィック著
野下浩平，星守，佐藤創，田口東 共訳
近代科学社
2,800円（第1巻），3,200円（第2巻）

■評価チャート



訳書第3巻の刊行が待たれるアルゴリズムの名著

本書はアルゴリズムの名著の誉れ高い Robert Sedgewick〝Algorithms, 2nd Ed.〟(Addison-Wesley)の翻訳である．原書は1冊であるが，この翻訳は3分冊になっている．

この本の紹介は，この連載を始めたころにやろうと思っていた．第1巻がそのころ（'90年10月）に出たからである．ところが第2巻が出たのは，'92年2月であり1年半も経ってからだ．そして第3巻はまだ出ていない．しびれを切らしての見切り発車となった．

セジウィック先生は，計算機界の大御所クヌース先生の弟子にあたり，アルゴリズムに造詣の深い人である．クヌース先生がアルゴリズムを網羅しようとした大著“The Art of Computer Programming”(邦訳はサイエンス社から分冊で出ている）は，原書でも未完に終わりそうなので，「計算機界のスターウォーズ」と私は呼んでいる．一方，セジウィック先生が書いた本書は，原書はとうに出版が終わっているのに，翻訳がまだ完結していない．

翻訳を分冊にするのはよしとしても，それが1度に出ず，何か月も間隔をおいてから出ることへの不満は以前にも述べた．中には続きが何年経っても出ず，宙ぶらりんになっているものもある．けしからんことだ．せっかくの名著もこんな翻訳の都合によって，評価が下がってしまう．残念なことである．

クヌース先生の“The Art of Computer Programming”がアルゴリズムの百科事典だとすれば，本書はアルゴリズムの入門書である．入門書といっても巷によくある入門書とは違い，土台がしっかりしている．

アルゴリズムの性質を綿密に述べ，どういう状況でどのアルゴリズムを使うべきかの指針を示してある．後学のための参考文献紹介も豊富である．扱っているアルゴリズムは，基本的なものが多いが，高度なものも含まれる．アルゴリズムの記述にはPascalが使われている．アルゴリズム実行中のデータの変化を示す図も豊富で，理解を助けている．

訳書第1巻は，「基礎」と「整列」に大きく分かれる．「基礎」では，基本データ構造である配列，リスト，スタック，キュー，木について述べ，計算量を考えるなどといったアルゴリズム解析の基本を提示する．

「整列」はアルゴリズムの本では，定番の話題だが，「基数整列法（radix sort)」を詳しく解説してある点が目を引いた．

訳書第2巻は，「探索」，「文字列処理」，「計算幾何」の話題である．

「探索」では，初歩的な探索技術である逐次探索，2分探索のほか，平衡木，ハッシュ，基数探索を章を設けて解説している．外部探索では，索引順アクセス法(ISAM)，B木，拡張可能ハッシュ法を取り上げている．

「文字列処理」では，文字列探索の章で，ボイヤーームーア法の紹介もある．ファイル圧縮や暗号についても章を設けてある．

「計算幾何」は，アルゴリズムの入門書ではあまり取り上げられない分野である．この部分は「計算幾何」へのよい入門になっている．評者は知らないことばかりだった．

ワンラインNEWS
SGS・トムソン・マイクロエレクトロニクスはプログラムの書き換えが100万回できる4KビットEEPROMを発売した．

訳書第３巻は,「グラフのアルゴリズム」,「数理アルゴリズム」,「トピックス」に分かれる.

「グラフのアルゴリズム」では，グラフ探索などを,「数理アルゴリズム」では，乱数などを扱っている.「トピックス」で取り上げられている話題は，並列アルゴリズム，動的計画法，ＮＰ完全問題などである．難しいものばかりなので，概要紹介にとどめてある．

なお，本書のアルゴリズムをCで書き直した“Algorithms in C”(Addison-Wesley）も原書では出版されている．Ｃに親しんでいるプログラマは，こちらの方がいいかもしれない．

普段何気なく使っているアルゴリズムは，果たしてどういう性質を持っていて，そのアルゴリズムを使うのは適切だったのかどうか，といったことを見直してみるにも本書は役立つ．

早く訳書第３巻が出版されることを望む．

国内発売が待たれる「CC:Mail」，一足先に関連書を紹介

古田島義和（神田・書泉グランデ）

ロータス社の「CC:Mail」は，日本ではまだなじみのないことばでしょう．これは，全米市場で約30%のマーケットシェアを占め，インストールベースでも170万以上のユーザー実績をもつ米国No. 1のPC LAN対応電子メールソフトです．CC:Mailは，使いやすさと豊富な機能とすぐれたユーザーインターフェースを備えているばかりでなく，接続性を重視した設計となっており，MS-DOS, Windows, OS/2, Macintoshの各OSに対応したシステム管理用ソフトウェアとユーザーソフトウェアを提供しています．将来は，OpenLook, OSF/Motif, PMなどにも対応する予定です．MS - Network, LAN Manager, NetWareなどの主要ネットワークOSに対応しています．現在サポートされているトランスポートサービスは，非同時，X. 25, TCP/IP, PBXなどで，ゲートウェイはNotes, X. 400, FAX, SMTP, UUCP, SoftSwitch, 3Com, Banyan, MHS, SANDS/D/SOSS, PROFS, AS/400 office, All-in-One, VMSメール, Wong VS, HP Desk, EasyLink, SprintMail, MCI Mail, Geiscoです．計画中のトランスポートサービスおよびゲートウェイには，SNA LU6. 2/PU2. 1, Novell IPX/SPX, Apple OCE接続などがあります．

メール管理では，ユーザー名／パスワード，多元的暗号化など強力なセキュリティ機能を備え，集中管理ツールには，トラフィック，性能監視，集配信レポート，ユーザー登録・削除，メール・リスト，メールボックス監視などの管理システムがあり，ロータス社の製品群はもちろんのこと，Apple, IBM, Novell, Borland, Microsoft, ユーザー・アプリケーション, ISVアプリケーションに対応しています．ロータス社は日本での発売を年内中に予定しています．このソフトの普及が楽しみです．

解説書はすでに，Prentice Hall(M&T)社の「Delivering CC:Mail」(E. Armum著)が入荷しています．今後は同社の「Using CC:Mail」(S. Caswell著)，同(Howard W. Sams)，「The First Book of CC:Mail」(D. Derrick著)，「LAN Desktop Guide to E-mail with CC:Mail」（B. Fryer著）などが入荷する予定になっています．

最後に洋書の新刊として，Prentice Hall社（Howard W. Sams）の数点を紹介したいと思います．UNIX Desktop Guide toシリーズとして，「Open Look」(W. Barkakati著)，「X/Motif」(同著)，「Korn Shell」(J. Valley著)，「Emacs」(R. Roberts著)，「Tools」(B. Holliker著)，「UUCP」(J. Valley著)，「C/C++」(M. Newcomb著)，LAN Desktop Guide toシリーズとして，「Security with Netware」(Sams著)，「Disaster Recovery」(P. A. Cochlin著)，「Trouble shooting with Netware」(R. Segal著)，「Printing with Netware」(Sams著)が発刊されます．先ほど紹介した「Email with CC:Mail」(B. Fryer著)など，ガイド書としては最適なシリーズです．

また，新版になって入荷したM. I. T. Pressの「New Hacker's Dictionary」が好調で，品切れ間近となっています．

■和書&洋書ベスト10　　（書泉グランデ調べ：1992年３月１日～３月31日）

	和書	出版社		洋書	出版社
1	こんなもんいかがっすかぁ	アスキー	1	Modern Operating Systems	Prentice Hall
2	MS-DOSテキストデータ料理学	翔泳社	2	Power Programming with RPC	O'Reilly
3	初めて使うGNU Emacs	啓学出版	3	Writing Windows Device Drivers	Addison Wesley
4	オブジェクト指向狂詩曲	技術評論社	4	C++ Programming Language 2/E	Addison Wesley
5	GNUツールボックス	ソフトバンク	5	Standard C Library	Prentice Hall
6	パソコンLANとは何か	講談社	6	Mac Sheme	M.I.T.
7	アドベンチャーツクール98	アスキー	7	Mac Math	Springer-Verlag
8	宇宙の旅	アスキー	8	Learning GNU Emacs	O'Reilly
9	MINIXを256倍使うための本	アスキー	9	Metafont Book	Addison Wesley
10	カオス―新しい科学をつくる	新潮社	10	Object-Oriented Modeling & Design	Prentice Hall

★　★　★佐々木哲の★　★　★

U.S.プロダクツウォッチ

★安定性を増してMS-Windows3.1登場

USで4月6日，MS-Windows3.1が発表されて程なく，日本でも英語版，IBM互換機用のものがそのまま手に入るようになりました．英語版のパッケージのまま買うと，中に日本のマイクロソフトへの登録葉書が入っているのはご愛敬．

これは発売から2年たったMS-Windows 3.0の多くの不具合を解決し，より安定し，完璧なものへと近付けたバージョンアップです．互換性を保つために細心の注意が払われ，ほとんどの旧来のアプリケーションはそのまま使うことができます．ただし，ディスプレイドライバなどは，新たにサポートされたTrueTypeのために3.1対応のものに入れ換える必要があります．

あまり注目されないところでは，RS-232Cドライバが大幅に改良されています．以前この欄でもお知らせしたとおり，MS-Windows 3.0版ではNS16550などすぐれたバッファ付UARTがサポートされていなかったり不満がありましたが，今度のものはMS-Windows上でのDOS通信プログラムを含め16550がサポートされています．これによって，マルチタスク環境下でも安定して高速な通信が可能になりました．

★ウィルスは実在する！　安全ではない日本

マスコミや噂の中にしか存在しないと思っているあなた，ウィルスは実在します．そして確実に広がっています．今年のはじめミケランジェロウィルスが話題になりましたが，これは氷山の一角に過ぎません．

USではFBIが警告を出すほどの騒ぎになりましたが，日本のほうがUSよりもウィルス感染率は高いと思われます．私自身が昨年夏までアメリカに滞在していた時には，ウィルスなどまったく見たことがありませんでした．その間にも日本に帰った際など数件，自分のところ，顧客のところとウィルスを見つけています．

これはUSでは，ユーザー同士がFDを直接コピーしたり，プログラムをコピーするチャンスが少ないこと（東京のように密集してPCユーザーがいるわけではないですから），またローカルのBBSでも必ずといってよいほど，アップロードされたファイルは自動的にウィルスチェックが行われているためでしょう．

一方，日本はウィルスの巣である香港，台湾などと比較的近く，また無防備であることからこれだけの蔓延を招いていると考えられます．

一番のウィルス感染の防御は，McAfee AssociatesのVIRUSCAN*1を入手し，定期的にあるいは新しいプログラムを入手した時にチェックすることです．このコラムの読者の方は，もちろんすでに実行されていることでしょうが，日本でもパソコン通信などによりこのシェアウェア，SCANを容易に手にいれることができます．既存のウィルスの“指紋”を探す仕掛のものです．次々に現れる新手のウィルスに対抗すべく頻繁にアップデートされています．実際に私がこの2月に見つけたものは，SCANででもほんの数か月前から見つけられるようになったばかりのもので，アップデートを怠っていた古いものでは見つけられませんでした．

最新のSCANでは500種類近く，1000種近くの変種まで見つけることができます．SCANを走らせて感染していることが見つかったらどうするか．すぐにPCを使うことを止め，ライトプロテクトされたオリジナルのDOSのFDで立ち上げ直します．これはほとんどのウィルスが感染後，自動的にメモリに常駐しているためです．その後，もし運がよければ，同じくMcAfee AssociatesのウィルスCLEANで，感染したプログラムあるいはブートセクタなどからウィルスを取り除くことができます．あるいは，オリジナルのプログラムをインストールし直すことになります．ウィルスに感染したからといってすべてディスク内のプログラムを消去したり，ディスクをフォーマットし直したりする必要はありません．伝染病にかかったからといって，その人の家を焼いてしまうことはないのです．このCLEANもSCANにあわせてバージョンアップされています．定期的に入手しておいたほうがいいでしょう．

ここまでの話はIBM PC互換機でのことです．困ったのが同じDOSでありながら互換性のない国産の機種です．ウィルス自体はこれらのプログラムに感染したりしますが，誤動作しますし，ましてやこれらの除去プ

ワンラインNEWS
ソニーは「NEWS」シリーズの最高速モデルとサーバ機を発売．米ミップス社製のRISC型MPU「R4000」を搭載，処理速度は100MIPSとWSでは最高速を誇る．

ログラムがうまく動作しません。それだけに注意が必要です。

＊1：VIRUSCAN　$25
CLEAN-UP　$35
McAfee Associates
1900 Wyatt Drive, Suite 8
Santa Clara, CA　95054-1529
U.S.A.
phone (408)988-3832
fax (408)970-9727
(408)988-4004 BBS 2400 bps
(408)988-5138 BBS HST 9600
(408)988-5190 BBS v32 9600
CompuServe GO VIRUSFORUM

★高速なLocal bus

高速なグラフィックディスプレイボードなどが要求されるようになり，バスの転送速度のボトルネックが問題になってきています。ET4000に代表される高速VGAボードでは表示メモリへのアクセス速度がメインメモリとほとんど変わらないほどになってきています。それだけにISAバスという規格ではついていけなくなりました。EISAにしても，やはりバスとして負荷はあります。ましてやEISAマシン自体一部のサーバ用としてしか今のところ存在価値がない上に，高価で複雑なEISAバスのグラフィックボードは，Mylexなど一部からしか現れそうにありません。

今急激に脚光を浴びているのがLocal busです。要はさまざまなデバイスを繋ぐことを考えた複雑なバスを省略し，高速で安価なCPUからの切口を作ろうというアイデアです。

新しいだけに標準のないこのLocal bus，昨年秋ごろからOrchidやMicronics[*2]は独自のものをつけたマザーボードを出してきています。これではメーカーの用意したグラフィックボードしか使えず，すぐ陳腐化してしまいます。そこで，Video Electronics Standards Association (VESA)がLocal busの標準化に乗り出しました。この春のComdexでは早くもMicronicsがこのVESAで検討中の規格のLocal busをのせたマザーボードを展示しています。

この32ビットのバスでは，ISAの約16倍の速さでデータが転送できるそうです。まだVESAで最終的に決定されていない規格ですが，これに対応したグラフィックカードやメモリ，ディスクコントローラ，LANカードなどの登場が期待されます。

たとえば，Alpha Research Corp は，このVESA Local busに対応したキャッシュ付の高速IDE HDコントローラを発表しています。定価は700ドルから800ドル程度のこのコントローラにより，12から19Mバイト/sectという高速の転送速度が期待されるそうです。このVESAの規格は，来る6月のPC Expoまでには決まるようです。そのころには，さまざまなLocal busボードを見ることができるでしょう。

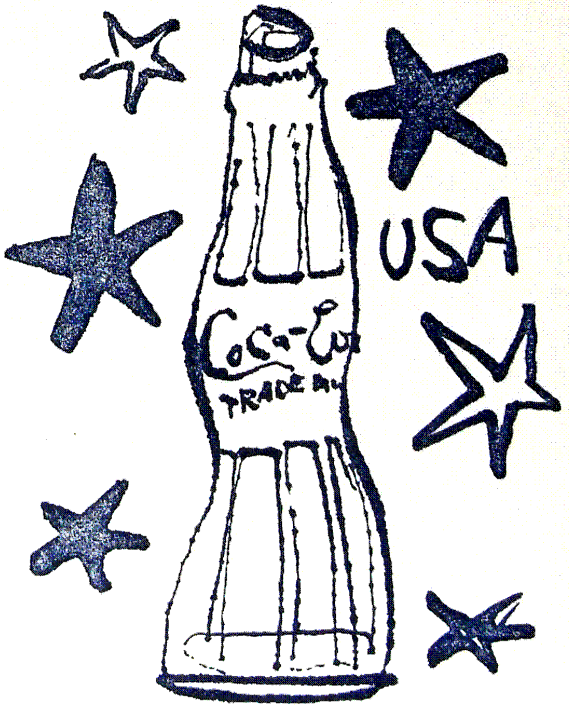
このまだ決まらないバス規格に対応すべく，CMS Enhancements Inc.はAnybus（何のバスでも）[*3]という名前のラインのアップグレード可能なPCを発表しています。486SX CPUからより上位のCPUにアップグレードできるだけでなく，将来のLocal busに対応できるSuperSlotが装備されているとか……。

＊2：Micronics Computers Inc.
Fremont, California
phone (415)651-2300
＊3：Anybus
486SX 20MHz 4MB RAM 40MB HD $1,399
CMS Enhancements Inc.
phone (714)222-6000

★今月のグラフィックボード

Genoa[*4]は，IBM/SGS ThompsonのXGAチップを使ったマイクロチャネル用XGAカードを発表しました。Super XGA 9000という名前で，オリジナルのIBMのものよりいくつかの点で優れているようです。1024×768の解像度を72Hzのノン・インターレースで表示できるだけでなく，表示速度がIBMのものより20%ほど速いそうです。

＊4：Genoa SuperXGA 9600 $799



Illustration：Masami Nozue

Illustration : NOBUKI FUKUHARA

西川和久の

希望的観測

Windows 3.1ついに出荷開始!!
Windows 3.0との違いとは??

1 遅れ遅れて……チューンナップされて登場

1992年4月6日，去年から噂されていたWindows 3.1がやっと出荷開始となった．高速化，安定化，TrueType，マルチメディア対応など，0.1のバージョンアップなので，一見地味なものと思われがちであるが，実はWindows 3.0と比べるとかなり細かくチューンナップされている．また，先の項目に隠れて一般的にあまり表に出てこないが，リアルモードを外してしまったのも大きな要素である．

リアルモードは，そもそも見かけがWindows 3.0で，中身はWindows 2.11のシステムで，旧アプリケーションや80286未満のCPU救済策として作られた．ところが，蓋を開けてみるとCPUの進歩やWindows 3.0にアプリケーション側が対応するペースが予定以上に速かったらしく，結果，SWAPFILE.EXEを起動する時以外はまったく使われない，無用の長物だったのだ．その上，Windowsのシステム側から見ると，リアルモード/標準/386拡張モード用の異なる3つのカーネル，ドライバ内での作動モード切り分けなど，管理が面倒なだけでメリットは何もない．1つモードが減ったことにより，システム自体はずいぶん軽くなったといえよう．

2 高速化，安定化 UAEを減らし，ディスク管理も自前で

Macの爆弾に相当するUAE……これをできるだけ少なくして，システムの安定化をはかるのが第一課題であった．実際に使ったところ，この目標はかなりのレベルまで達成できているようだ．MVDMについても[Ctrl]＋[Alt]＋[Del]でセッションごとにリセットをかけることが可能になり，万一，DOSアプリケーションがハングアップしても安心である．

また，これまでWindowsシステム中で唯一DOSファンクションを使用していたディスク周りも，386拡張モードではFastDiskと呼ばれるディスクBIOSをダイレクトにアクセスする機構が組み込まれた．面白いことに，以前OE (Operating Environment) と呼ばれていたWindowsも，Windows 3.1ではOS (Operating System) と呼び方を変えた．DOSを使うのはWindowsの起動とFATシステムだけになってしまったからであろう．

付属するDiskCache，SMARTDRVもStage Writeが標準設定となり，ファイルアクセス関係はかなり高速化され，描画関係も全体的に30%ほどアップし，全体的に磨きがかかった．昔からマイクロソフト社のOSやソフトウェアなどは，バージョンが3.1以上になって初めて普及期をむかえているが，Windowsも例外ではないようである．

ワンラインNEWS
横河・ヒューレット・パッカードはアップルコンピュータの「Macintosh」シリーズ用モノクロスキャナ「HP ScanjetⅡp」を発売．300dpiの読取解像度をもつ．

3 TrueType，マルチメディア この２つが売り！

今回の大きな売りはこの２点である．WYSIWYGを可能にするTrueTypeは，スペックを発表した当初「ATMがあるのに何を今さら……」などという意見が多かったが，Macでもサポートされたために急激に方向が変わってしまった．ATMより描画も速く，書体も数多く付属して，ユーザーにとっては嬉しいことである．

これまでMMEは，Windowsの拡張モジュール扱いであったが，Windows 3.1では標準機能となったため，アプリケーション側の対応も増えるであろう．余談になるが，私はMMEを日本語Windows 3.0に組み込んで，つね日ごろ使っている．パソコン上で，ビープ音でなくサンプリング・サウンドが鳴るとそれなりに楽しい．ただ社内で変な音がして，騒がしくなるため周りの人間には変な顔をされてしまうのが欠点である……(^^;

4 しかし互換性に問題も…… そして日本語版は!?

最近，海外のパソコン雑誌はWindows 3.1紹介記事オンパレードである．これまで説明したようによいところがいろいろある反面，Windows 3.0用アプリケーションとの互換性の問題が指摘されている．Windows 3.1だと不都合が出るアプリケーションがあるのだ．これは恐い話である．アプリケーションリストを眺めていて笑ったのが，なんとマイクロソフト純正の商品であるMS-Word 1.2が入っていたのだ!! すでに，新しいバージョンである2.0が販売されているため，マイクロソフト側はバージョンアップするよう，ユーザーに呼びかけているのだが，マイクロソフト純正でも問題が出るのだから，他社では頭が痛いところであろう．当然，日本語版Windows 3.1でもありうる話なので若干心配である．

日本語Windows 3.0で，ほとんどの試練（?）を乗り越えているため，日本語Windows 3.1は漢字TrueTypeさえ完成すれば，ほかはそれほど問題ないハズだ．しかし細かいところで英語版と仕様が異なりそうな雰囲気がある．たとえば，「英語版ではDOS窓のフォントサイズがいろいろ変更できるが，日本語版ではどうなるか!?」「MME関連のモジュールが落とされないか!?」など……不安と期待で複雑な心境だが，一刻も早く日本語Windows 3.1が出荷されて欲しい!!

そうそう話はまったく変わるが，今日４月16日の新聞にWindowsを著作権の侵害で訴えていたアップル社が，訴えを退けた記事が載っていた．ゼロックス社から横やりが入ったのか!?ことの真意はわからないが，これは意外である．また文中に「これでマイクロソフトが恐いのは独占禁止法だけになった」とまであった．笑えない話である．

コンピュータの法律

コンピュータプログラムの登録制度について・その2

熊谷信太郎(弁護士)

■創作年月日を登録する

コンピュータプログラムを新たに作成すると，その時点で何らの手続きを要せずにプログラムの著作権が発生します．しかし，プログラムの著作権が生じたからといって，それだけで安心しているわけにはいきません．

他社が著作権の盗用を行い，非常に似ているプログラムを発売した場合でも，相手が盗作であることを否定した場合には，相手より先にそのプログラムを製作していることを証明して，プログラムの著作権を有することを明らかにしなければ，著作権侵害を理由に他社のプログラムの利用や発売を差し止め，損害賠償請求をすることはできません．人より先にプログラムを作成したことを立証した者が，ほかの者に対して同一内容のプログラムの使用は著作権侵害であると主張できるのです．

ところが，プログラムの中には公表されずに密室で使用されるものもあるので，プログラムの創作年月日を裁判で立証することはそう簡単ではありません．そこで，そのような場合に備えて，コンピュータプログラムの創作年月日を登録することが認められています．その登録をしておけば，簡単に創作年月日の立証ができるので，たいへん便利です．

また，前回説明したように，プログラム著作権を買ったところ，売主が同じプログラム著作権を別の買主に二重に譲渡してしまった場合にも，プログラム著作権の登録が権利取得の対抗要件とされていますから，どちらの買主が先にプログラムを買ったかではなく，どちらの買主が先にプログラム著作権を登録したかによって，著作権取得者が決まります．

■対抗要件としての著作権登録

対抗要件という法概念は，なかなか理解しにくいかもしれませんが，権利を所得するための要件ではなく，権利を取得した後に，その権利を他人に主張するための要件という意味です．したがって，自分だけで使用するためのプログラムを作成したような場合，その時点で著作権が発生していますから，それを他人に譲渡したり，質権を設定したりしない限り，登録をしなくても困ることはほとんどありません．

この対抗要件という概念はコンピュータのプログラム著作権に特有のものではなく，たとえば土地などの不動産についても観念しうるものであり，土地についての対抗要件は登記です．土地の所有権は登記しなくても取得することはできますが，登記しなければ，取得した所有権を第三者に対抗することはできないのです．

また，プログラム著作権も財産権ですから，それを担保に質権を設定して，金融期間から融資を得ることができますが，そのようなプログラム著作権に対する質権の設定，移転，変更，消滅などについても，著作権の登録が対抗要件になります．

■実際の登録方法

このようなコンピュータプログラムなどの著作権の登録については，文化庁が所管しており，文化庁が管理する著作権の登録原簿（コンピュータプログラムの場合には原簿といっても実際には磁気テープを使うことになっています）に記載することによって効果が発生します．

プログラムの登録に際しては，マイクロフィルム（日本工業規格A6版マイクロフィッシュ）に複製して提出する必要がありますが，登録できるプログラムはOS，アプリケーションプログラムなど，その種類，用途，使用言語を問いません．

コンピュータプログラムの実際の登録手続きは，文化庁の外郭団体である財団法人ソフトウエア情報センター（略称：SOFTIC）*[1]が行っています．

登録には，登録手数料が20,000円と登録免許税（3000円～18,000円くらい）がかかります．

登録する場合に，気をつけなければならないのは，すでに同じようなプログラムが開発されて登録済みの場合に

ワンラインNEWS
ソニーはCD ROMソフト作成用のCASEツール「ディスクシーンNWF-644」と，シミュレーションボード「NWB 261」を発売した．いずれも同社のWS「NEWS」上で動作する．

は，プログラムの登録をしても無駄になってしまいますから，現在までのところどのようなプログラムが登録されているかを調査する必要があることです．登録原簿の調査のための手続きも，SOFTICで行っています．調査の際には，登録原簿そのものの閲覧や登録原簿の謄抄本の交付はできませんが，それにかわるものとして，登録事項記載書類の交付を求めることができますので，それを調べて，登録するかどうかを決めることになります．

＊1：（財）ソフトウエア情報センター
〒105 東京都港区虎ノ門
5丁目1番4号
東都ビル4階
TEL 03-3437-3071

アキバ短信

ビルトイン

「情報を提供できる店」を目指すDOS/Vショップ

昨年から今年にかけてパソコン雑誌などでは，DOS/Vに関する記事が多くみられた．出版物だけを見ていると，DOS/Vが一大ブームを巻き起こしているかのようだが，実際にDOS/Vマシンがどれだけ売れているかはわかりにくい．少なくとも，日本のパソコンのシェアが塗りかえられるほどの台数が出ていないのは確かだ．それというのも，DOS/Vのプロモーションに力をいれているのは，パソコン大手メーカーでは日本IBMだけで，あとは海外のパソコンメーカーと提携した販売会社ががんばっているのみというのが実状だからだ．

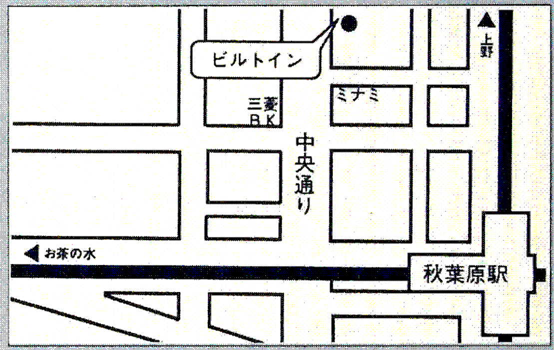
今回ご紹介するビルトインも台湾のメーカーと組んで，PC/AT互換機を販売するDOS/Vショップだ．昨年11月に開店して以来，徐々に店頭の品ぞろえが整ってきつつあるところだ．

「初心者のお客様が思ったより多く来られるのでびっくりしています」と，話すのはビルトインを経営する（株）クラスターの森田社長．「雑誌で紹介されるのならぜひ読者の方に伝えていただきたいのですが，ATマシンは98とは違うということを，くれぐれも理解していただきたいですね．ATは世界中でさまざまなものがつくられています．マシン本体，ボード類など，同じチップが載ったものでもピンキリだということなんです．間違ったものを入手してしまうと動かないこともあります．」

「2，3年後をめざして土台作りをするためにはじめた店なので，最初はマニアのお客様ばかりだろうと思っていたのですが，初心者の方が多くて……」と続けるのは，店長の田中氏．「はっきりいって，初心者はもう少し様子を見てほしいですね．大手の互換機メーカーがしっかりしたサポート体制で参入してくれれば，初心者でも安心して使えるようになると思います．」多くのDOS/V入門者はWindowsを動かすことを目的として購入するため，「スーパーVGAで動かしたいのに640×480でしか動かない」など，ビデオまわりのトラブルが多いのだそうだ．「ATは完成品といってもキットのような側面を持っています．現在のDOS/VはDOSを理解するだけでなく，ATのハードのことも少しはわからないと使えません」（田中店長）．

ビルトインで販売しているマシンは，台湾製のEFA（イーファ）．売れているマシンは，やはり7割ほどが486だという．VGAボード，内蔵ハードディスク・FDドライブなどPCのお店としては細かな品ぞろえだ．キャッシュコントローラーボードは，ビルトインがはじめて取り扱ったとか．オリジナルのストリーマーを発売するなど製品開発も熱心に行っている．「ものを売るだけでなく，情報を提供できるショップに」という田中店長の言葉に期待したい．

営業時間　12:00～19:00（平日）11:00～19:00（日祝日）
定休日　木曜日
TEL　03-3257-3264



●プログラマの仕事場　15

株式会社デジタル・リサーチ・ジャパン

DR DOS/Vで再び一般ユーザーの前に登場するOSメーカー

コンピュータの歴史では，10年前はすでに神話の世界だ．CP/M栄光の日々，IBM-PCの誕生，MS-DOSの成功物語などが，さまざまなエピソードに彩られて神話の世界に納まっている．デジタル・リサーチ社も創世神話の主役であったが，80年代後半は一般ユーザーの前にはあまり登場しなかった．そのデジタル・リサーチ社が，いままた現実の世界でアクティブな役まわりを獲得しようとしている．

FlexOSはPOSシステムなどで活躍

CP/Mは，マイクロコンピュータ草創期に普及した8ビットマシンのオペレーティングシステムとして重要な役割をはたした．そのCP/Mを開発したのが米国デジタル・リサーチ社であり，今回訪ねたデジタル・リサーチ・ジャパンはその100%出資の日本法人である．CP/Mは，16ビットマシンが主流になって以降，表舞台からは姿を消したが，さまざまな制御システムなどに組み込まれて，現在も活躍している．一太郎でパソコンを始めたユーザーが大勢を占めている日本のパソコン状況では，CP/Mやデジタル・リサーチの名を知る人は少なくなってしまったが，CP/Mの組み込みシステムは案外生活のさまざまな場面で活躍しているのである．

デジタル・リサーチは，CP/Mからの伝統を受け継いで組み込み型OSを主力として活躍してきた．現在，同社の製品は3系統に分かれている．ひとつが，DOS互換のマルチタスクOSであるConcurrent DOS，次がリアルタイムOSであるFlexOS，そしてDOS互換のパソコンOSとして知られるDR DOSである．

このうちリアルタイム処理のFlex-OSは，POSシステムなどにおいて有力なOSとして，さまざまなシステムに採用されている．POSシステムでは，販売時に料金計算をすると同時にその情報が集計センターに送られ，全社的な商品管理，販売管理などが可能となるシステムである．そこでは，実時間での処理が要求されると同時に，処理の正確さが求められる．マルチタスク，マルチユーザー環境でリアルタイム処理を実現するためにCPU性能を最大限に引き出さなければならない．そうした要求を満たすOSとして，FlexOSはさまざまなシステムに採用されている．リアルタイムOSは，POSシステムだけでなく，ファクトリオートメーションやビル管理，銀行業務など，さまざまなマーケットで活躍している．

組み込みがDOSになるのは開発側の事情も

日本では，組み込みシステムの需要が多いという．同社汎用OS技術部長の

ワンラインNEWS
兼松は米クオリティ・セミコンダクタ（QSI）と日本での販売代理店契約を結んだ．QSIは超高速スイッチやSRAMなどが主力で，WSや高級パソコン向けに売り込む．

崎田達郎氏は，「アメリカでは，レジなどにもPCを置いているところが多いのですが，日本ではパソコンの形をしていたのでは敬遠されるようですね」と話す．日本語ワープロ専用機が成功を納めているように，日本のマーケットでは専用機が受け入れられやすいのかもしれない．

また，米国でDOS互換OSとして成功しているDR DOSもはじめからROM化できるように開発されている．実際，ROM化された組み込みシステムの需要が多くなっている．組み込み用のDR DOSは，Embedded DR DOS（DOS/E）と呼ばれる．

「組み込み型のさまざまな機器がDOSを必要としているという事情もあります．たとえば，ファイルシステムを使わなければならない場合などですね．一方で，わざわざDOSを使わなくてもいいのではないかという場合もあります．しかし，これだけDOS環境が普及しているわけですから，開発効率という面からDOSを採用していることもあるのではないでしょうか．ツールなどが豊富な使い慣れた環境でソフトウェアを開発できるというメリットがあります．機械語で開発できる人材を養成しなくても，技術者が確保できるということもあるでしょうね」と，崎田氏は組み込み型DOSシステムが普及する理由を教えてくれた．

汎用OS技術部長　崎田達郎氏

DOSの規模が1人でできる限界

デジタル・リサーチ・ジャパンが発足したのは1983年のこと．以来，米国カリフォルニア州モントレーの本社やイギリスにあるヨーロッパの拠点であるEDC（ヨーロピアン・デベロップメント・センター）と連携して活動している．現在もモントレーとEDCに各1名が駐在している．

ソフトウェアの日本語化に関しても，基本的なDBCS対応などについては海外で行う．日本で開発するのは，デバイスドライバなどに限られているという．

デジタル・リサーチ・ジャパンのエンジニアは約10名で，汎用OSグループとリアルタイムのFlexOSグループに分かれている．

About Company

株式会社デジタル・リサーチ・ジャパン．米国デジタル・リサーチ社の全額出資による日本法人．米国デジタル・リサーチ社は，ゲリー・キンドールにより1976年に設立された．同年に発売されたオペレーティングシステムCP/Mは，8ビットCPUの標準的OSとしての地位を獲得した．日本法人は，1983年に設立され，OS製品のOEM，VARを中心とした事業展開をしてきた．昨年，米国本社はノベル社と合併し，ノベル社の1部門として活動している．日本においてノベル社との関連をどうするかは未定で，当面はデジタル・リサーチ・ジャパンとして活動する．

Location

東京・五反田のTOCに近いビルに事務所を構える．テストルームには，同社の歴史を物語るように，10年近く前のものと思われるマシンも見られる．

Products

各種オペレーティングシステムがおもな製品．CP/M-80，CP/M-86，Concurrent DOS 86，Concurrent DOS XM，Concurrent DOS 386，FlexOS 186，FlexOS 286，FlexOS 386，DR DOSといった製品群を用意している．

現在，パソコン用OSとして注目されているDR DOSは，昨年夏に欧米で6.0が発売され成功を納めている．このたび，日本で発表されたDOS/V版は，このDR DOS 6.0を日本語対応にしたもの．DR DOSは，DOS互換であることに加えて，さまざまなユーティリティを用意することでユーザーの支持を得てきた．DR DOS 6.0の特徴的なユーティリティを紹介しておこう．

・DiskMAX

ハードディスクの容量とパフォーマンスを向上させる．データを2分の1に圧縮してハードディスクに書き込み，読み込みの時に伸長する．ディスクキャッシュ，ディスクの最適化機能なども持つ．

・TaskMAX
20個までのタスクをロードし切り替える機能を持つ．

・MemoryMAX
286システムでは，カーネルなどをハイメモリに，ドライバとTSRをアッパメモリにロードする．386システムでは，カーネルなどをハイメモリにロードするほか，Windows3.0の各モードのサポート，LIMエミュレーション機能を持つ．

・ViewMAX
グラフィカルなコマンドインターフェース．タスク切り替えをサポートする．
このほか，ファイル転送，セキュリティなどさまざまな機能がある．

新旧のマシンがずらりと並んでいる

崎田氏自身は，もともとは汎用コンピュータのOSに携わっていた技術者だ．VM(バーチャルマシン)の入出力関係をやっていたという．その崎田氏が，マイクロコンピュータに関心を持つようなったのは，マイコンブームの時代にテレビで紹介された日立のマイコンを見てからだ．やがて，CP/MやDOSといった小型コンピュータのOSに携わることとなる．

技術者にとって魅力があるというUNIXだが，崎田氏はそうでもないという．「UNIXは，結局はポーティングやドライバの話になってしまうのです．やはり，OSそのものを作る立場でないとおもしろくありません．もうひとつは，DOS程度の規模が1人の人間がすべてを見ることができる限界でないかと思います」というように，優れているといわれるOSを扱っても，それがポーティングやOSの一部であっては面白味がないというわけだ．

柔軟なライセンス形態で DR DOS/V登場

デジタル・リサーチ・ジャパンは，米国およびヨーロッパで成功を納めたDR DOSをもって，パソコンOSの新しい展開をねらっている．一般のパソコンユーザーの市場にDR DOS/Vによって登場しようとしている．

日本IBMの提唱したDOS/Vは，IBM DOS(MS-DOS)に拡張を加えて，VGAを搭載したIBM PC/ATまたはPS/2にハードウェアを追加することなく，ソフトウェアで日本語環境を実現するDOSである．DR DOS/Vは，IBMのDOS/Vと同様の考え方で，DOS互換のDR DOSをベースとしてソフトウェアによる日本語環境を実現する．

DR DOS/Vは，ハードウェアメーカーにOEM提供されるが，DOS/Vマシンですぐに動作する形で提供される，OADG・J-3100・AX・米国101キーボードをインプリメント済み，ライセンスの形態が柔軟であるなどの特色がある．

当面は，同社が直接リテール版を出荷する予定はない．これは，リテール版を出した場合にサポート体制を整備する必要があるためだが，DOS/Vマーケットが成長すれば，リテール版が非常に有効性を持つ市場になるであろう．また，こうしたシステムソフトウェアがハードウェアメーカー以外から販売されるようにならなければ，DOS/Vが成功したとはいえないだろう．DOS/Vが目指すのは，ユーザーが好きなハードウェアやソフトウェアを自由に組み合わせて使うことのできる状況であるはずだからだ．

米ディジタル・イクイップメントと米インテルは次世代スーパーコンピュータのソフト開発を共同で行うことで合意したと発表した．

●イベント

■5月20日(水)〜5月23日(土)
ビジネスショウ'92TOKYO
「オフィスイノベーション――より人間らしく快適に」をテーマに、事務処理・情報処理システム、ソフトウェアを展示.
会場：東京国際貿易センター（東京・晴海）
入場料：無料（ただし5月20日・21日の招待日は招待券が必要）
問い合わせ先：(社)日本経営協会
TEL 03-3403-1331

■6月1日(月)〜6月5日(金)
第五世代コンピュータ国際会議1992
1982年に開始された第五世代コンピュータ研究開発プロジェクトの研究開発成果発表，招待講演，テクニカルセッションが行われる.
会場：東京プリンスホテル（東京・港区）
登録費：110,000円（学生は10,000円）
問い合わせ先：(財)新世代コンピュータ技術開発機構FGCS'92事務局
TEL 03-3456-3195

■6月10(水)日〜6月12日(金)
ビジネスショウ'92OSAKA
会場：インテックス大阪（大阪市・住之江区）
入場料：無料（ただし6月10日・11日の招待日は招待券が必要）
問い合わせ先：(社)日本経営協会関西本部
TEL 06-443-6961

■6月10日(水)〜6月12日(金)
'92テレコム・ジャパン情報通信総合展
LAN，パソコン通信，ISDN，衛星通信サービスなどの情報通信分野の展示会.
会場：サンシャインシティコンベンションセンターTOKYO（東京・池袋）
入場料：1,000円
問い合わせ先：日本経済新聞社事業局総合事業部'92テレコム・ジャパン運営事務局
TEL 03-3243-9082

■6月17日(水)〜6月19日(金)
LAN Expo '92 in Kobe
LAN(ローカルエリアネットワーク)関連製品の総合展示会，セミナー，ワークショップも開催される.
会場：ワールド記念ホール（神戸ポートアイランドホール）
入場料：1,000円
問い合わせ先：(株)マーコム・インターナショナル
TEL 03-3403-8515

■6月24日(水)〜6月27日(土)
CIMジャパン'92
製造業のための情報戦略・統合生産システムの展示会，FNE '92 (Fais Networking Event) も同時開催され，工場自動化相互接続システム(FAIS)のデモンストレーションが行われる.
会場：日本コンベンションセンター（幕張メッセ）
入場料：5,000円
問い合わせ先：リードエグジビションジャパン(株)
TEL 03-3349-8501

●セミナー

Microsoft University

■5月25日(月)〜5月29日(金)
MS OS/2 Presentation Manager Applications Programming
MS OS/2 PMアプリケーション開発のための上級コース.
受講料：180,000円（5日間）

■6月8日(月)〜6月12日(金)
Microsoft Windows Programming Environment
Windowsアプリケーション開発のための入門コース.
受講料：180,000円（5日間）

■6月9日(火)〜6月12日(金)
MS OS/2 Programming Environment
MS OS/2 PMアプリケーション開発のための基本コース.
受講料：150,000円（4日間）

■6月15日(月)〜6月19日(金)
Microsoft Windows Applications Programming
Windowsアプリケーション開発のための上級コース.
受講料：180,000円（5日間）

■6月22日(月)〜6月23日(火)
C言語入門
Quick Cを使ったC言語の入門コース.
受講料：60,000円（2日間）

■6月24日(水)〜6月26日(金)
C言語応用
MS-C 6.0を使ってC言語の高度な技術を学ぶ実習中心のコース.
受講料：90,000円（3日間）

Microsoft Universityは，OS/2，Windowsアプリケーション開発者のために定期的に開催されているセミナーです.
問い合わせ先：富士ソフトウエア（株）
品川教育センター　TEL 03-5461-8296

NEWS & NEW PRODUCTS

マイクロソフト，英語版QC for Windows，Visual Basicを発売

マイクロソフト(株)（インフォメーションセンター：TEL 03-3363-1201）は，Windowsアプリケーション開発言語「Microsoft QuickC for Windows」（英語版）と「Microsoft Visual Basic」（英語版）を発売した．通常の英文マニュアルに加え，インストール方法からプログラミングの初歩までをやさしく解説した日本語スタートアップガイドと日本語Windows3.0A上で稼働するためのプログラムが標準添付されている．

「Microsoft QuickC for Windows」は，C言語を利用したWindowsベースのアプリケーションを効率的に作成する開発環境を提供，エディット，コンパイル，デバッグなどのプログラミングに必要な作業はすべてWindows上の統合環境で行うことができ，グラフィカルユーザーインターフェースの作成もかんたんに行うことができる．

「Microsoft Visual Basic」は，従来，Windowsアプリケーション開発に必須だったC言語の知識がなくても，グラフィカルデザインツールとイベントドリブンプログラミングを組み合わせることによって，かんたんにWindowsアプリケーションを作成することができる．グラフィカルユーザーインターフェースはツールボックスにあるビジュアルコントロールをフォームに貼り付けるだけで作成でき，サブルーチンのフレームも自動的に作成される．

■Microsoft QuickC for Windows

本体：CPUに80286以上を搭載し，Microsoft Windows 3.0以上，または日本語Microsoft Windows 3.0A以上（日本IBM社製の場合は3.02以上）で稼働するマシン／必要メモリ：386エンハンストモード，スタンダードモードとも本体メモリ640KB以上＋プロテクトメモリ3MB以上（リアルモードでは動作しない）／ハードディスク：空き約7MB／価格：32,000円

■Microsoft Visual Basic

本体・必要メモリ：同上／ハードディスク：空き最小約2MB，最大約5MB／価格：32,000円

マイクロフォーカス，日本語COBOL/2 Workbenchを発表

マイクロフォーカスジャパン（株）は，Micro Focus 日本語 COBOL/2 Workbench（以下Workbench）を発表した．Workbenchには，同社のCOBOLコンパイラ「日本語COBOL/2」とエディタ，Advanced ANIMATORなどの生産性向上ツール「Micro Focus 日本語 COBOL/2 Toolkit」が含まれている．また同時に，Workbenchと併用することにより，IBMメインフレームベースのアプリケーションをワークステーションやパーソナルコンピュータへダウンロードし，開発，維持管理，テストを行うことのできるアドオンツールとして，「Micro Focus 日本語 COBOL/2 Mainframe Development Environment」と「Micro Focus 日本語IMSVS86」も発表した．対応機種はIBM PS/55シリーズで，アドオンツールの併用にはIBM OS/2 J1.2が必要．販売は，同社の販売代理店である住商コンピューター（株）（TEL 03-5600-5076）が行う．

マルテックス，PASCALを拡張した日本語プログラムを発表

（株）マルテックス（TEL 03-3403-3420）は，プログラミング言語PASCALの仕様を拡張し，日本語による記述が可能なプログラミング言語，日本語プログラム「TAMAGO 1.2」を発表した．「TAMAGO」は，コマンドなどを普通の日本語にきわめて近い形で記述でき，初心者がプログラムの学習や構造化の基本を習得するのに適している．実行モジュール（コンパイラ，インタプリタ）は，パソコン通信によるダウンロードが可能で，実行モジュールに関してはコピーフリー（フリーソフトウェア）になっている．ただし，マニュアル，ソースプログラムについては，別途有償で配布する．また，

ワンラインNEWS
オーエスアイ・プラスは国際標準プロトコルである「OSI」に準拠したWS用通信ボード「μ-OSI/V」とミニコン用通信ボード「同Q」を発売した．

NEWS & NEW PRODUCTS

パソコン通信未加入者のためにディスクコピーサービスも行う．対応機種はPC-9801，DOS/V対応機種で，リリースは6月の予定．コピーサービス，マニュアル・ソースプログラムの価格は，実行モジュールが2,400円，マニュアルが2,000円，ソースプログラムが2,800円，マニュアルと実行モジュールが4,000円で，それぞれ別途送料がかかる．

マイクロウェア・システムズ，OS-9開発パッケージを発売

マイクロウェア・システムズ(株)（営業部：TEL 03-3257-9000）は，モトローラ社製のMVME167シングルボードコンピュータ上で，OS-9/68040リアルタイムオペレーティングシステムがすぐ動作するように設定した開発パッケージ「OS-9/167 Development Pak V2.4」を発売した．OS-9/680X0用アプリケーションの開発用，またはシステムインテグレーションやI/O拡張のためのドライバ開発のベースシステムとして利用でき，パッケージにはOS-968040リアルタイムオペレーティングシステムのほか，OS-9アプリケーション開発ツールセットが付属している．価格は600,000円（日本語版），550,000円（USA版）．

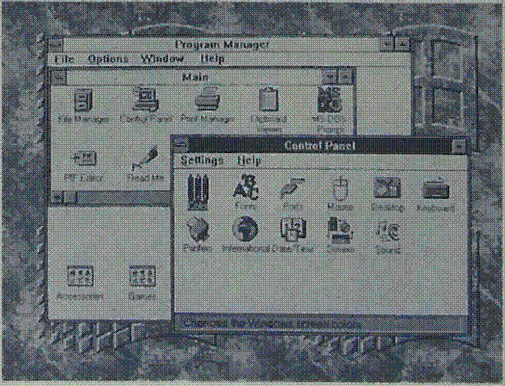
ティップス，98対応C言語用ライブラリを発売，バージョンアップ

（株）ティップス（TEL 03-5828-0511）は，PC-9800シリーズ対応のC言語ライブラリ「BGM/Kit」（39,800円）を発売，「CCF/総合コンソールコントロールライブラリ2.0」（59,800円），「BGM/グラフ作成プログラム2.0」（98,000円），「Window & Menu 2.0」（29,800円），「Key & Display 2.0」（29,800円）をバージョンアップした．全製品完全ソースコード付属，アプリケーションなどでの組み込みロイヤルティは無料となっている．

米マイクロソフト，Windows 3.1，Excel 4.0を発売

米国マイクロソフト社は，「Microsoft Windows 3.1」，「Microsoft Excel 4.0 for Windows」を発売し，そのおもな機能強化点を発表した．「Microsoft Windows 3.1」は，使いやすさ，処理速度，安定性などの面で多くの機能強化がなされた．

使いやすさの面では，登録ユーザー名と印字出力を行うプリンタ名を入力するだけで，ハードウェアの詳細を識別して適切な構成をセットアップする高速セットアップ機能，スタートアップグループへの登録により起動時にアプリケーションを自動実行する機能などをもつ．プログラム名の折り返し表示やスクリーンセイバーの提供など，グラフィカルユーザーインターフェース面での強化がなされ，練習用ソフトウェアも付属している．処理速度と安定性の面では，ファイルマネージャ，ディスクキャッシュの高速化を実現，フォントを動的にプリンタへダウンロードする機能やユニバーサル・プリンタドライバによる効率的なすばやい印字出力が可能となっている．このほか，MS-DOSアプリケーションのサポート強化，ノートブックパソコンなどのバッテリ駆動システムの電池持続時間を最長25%まで向上させるAPM（アドバ



Microsoft Windows 3.1

NEWS & NEW PRODUCTS

ンスト・パワー・マネジメント），スケーラブルフォントTrueType，OLE（オブジェクト・リンキング・アンド・エンベディング）ライブラリ，マルチメディア機能（音声制御・ビデオプレイヤー制御など）の搭載など，新機能も追加された．また，サーバへの接続状況を保持し，再起動時に自動接続を行うほか，LAN ManagerやNetWareへのアクセスを簡便化し，管理機能を容易に利用できるなどのネットワークへのサポート向上もなされた．日本国内での販売価格は24,000円．なお，日本語版の出荷時期については未定．

「Microsoft Excel 4.0 for Windows」は，新たに，ワークシート上のデータをマウスで直接つかんで移動することができる「ドラッグ／ドロップ」，また，たとえば"1月"と入力されたセルを横または縦方向に引き伸ばせば自動的に"2月""3月"と入力される「オートフィル」，複雑な操作を順に誘導してくれるアシスタント機能「ウィザード」などの機能が追加された．グラフ作成用，作図用など，作業ごとに分類された複数のツールバーを用意，オンラインヘルプも改善された．複数のシナリオを1枚のワークシートで分析，保存ができる「シナリオマネージャ」，Excel 3.0の作業状態ファイルを拡張し，1枚のワークブックファイルに複数のワークシートを結合することができる「ブック」のほか，140の関数の追加，印刷とプレゼンテーション機能などの強化がなされた．また，Lotus 1-2-3 Release 2.01互換のマクロを修正せずに利用することのできる「ロータスマクロインタプリタ」を備え，Lotus 1-2-3とImpressファイルなど19種類のファイル形式を直接読み書きすることができる．発売はWindows版が4月中旬，System 7.0に対応したMacintosh版は5月中旬の予定．なお，日本語版の発売は1992年後半の予定．

エー・アイ・ソフト，パーソナルUNIXシステム「PANIX」を発売

エー・アイ・ソフト（株）（TEL 0263-36-1424）は，本格的パーソナルUNIXシステム「PANIX」を発表した．カーネルには「UNIX System V/386 R4.0」を使用し，メッセージを日本語化することで，わかりやすい操作を実現した．インストールはメッセージに従ってフロッピーを入れ替えるだけで，ハードウェアに最適な環境を自動的に作成する．環境設定ソフト「Sysadm」，GUI環境として「X Window System Version11R4」を搭載し，プログラム開発環境として「GNU C Compiler V1.40」を提供している．発売開始は6月1日．限定5,000本に限り特別価格48,000円で直接販売を行う．対応機種は，NEC PC-9800RL，RS，RA，DA，FA，H98SとEPSON PC-386G，S，P，GS，GE．必要機器は，プロテクトメモリ5.6MB以上（推奨9.6MB），SCSIハードディスク100MB以上（推奨130MB）．

三菱電機，AX-VGA/H対応ノートブック型パソコンを発売

三菱電機（株）（情報処理機器事業部計画部：TEL 03-3218-3285）は，表示方式にAX-VGA仕様に準拠したAX-VGA/H方式を採用したノートブック型パソコンの新機種「MAXYNOTE386F」「同386E」を発売した．「MAXYNOTE386F」は，80386SLを搭載したジャストA4サイズのノートブック型パソコンで，16KBのキャッシュメモリ，最大8MBのメモリ搭載が可能．「同386E」は，80386SXを搭載したA4ファイルサイズ，普及型のノートブック型パソコンで，2FDDタイプも用意されている．

■MAXYNOTE386F（M3125-A14/A18）

プロセッサ：80386SL（20MHz）／メモリ：2MB（最大8MBまで拡張可能）／外部記憶装置：3.5インチFDD×1・40MB HDD×1（A14），3.5インチFDD×1・80MB HDD×1（A18）／ディスプレイ：冷陰極管バックライト付STN液晶ディスプレイ／価格：478,000円（A14），558,000円

ワンラインNEWS
横河・ヒューレット・パッカードはWS上でCD-ROM電子出版物を利用できる検索ソフト「HPLaser VIEW」を開発，発売した．同社のWS上で動作する．

NEWS & NEW PRODUCTS

(A18)

■MAXYNOTE386E (M3123-A20/A12/A14)

プロセッサ：80386SX (16MHz) ／メモリ：1MB (最大5MBまで拡張可能) ／外部記憶装置：3.5インチFDD×2 (A20), 3.5インチFDD×1・20MB HDD×1 (A12), 3.5インチFDD×1・40MB HDD×1 (A14) ／ディスプレイ：冷陰極管バックライト付STN液晶ディスプレイ／価格：268,000円 (A20), 338,000円 (A12), 378,000円 (A14)

金亀電子，PC/AT互換の個人向け高級機シリーズを発表

金亀電子 (株) (EUROTECH - AEGiS - HOTLINE：TEL 03-5256-7658) は，同社が販売代理店となっているユーロテック社製IBM PC/AT互換機の個人向け高級機「Rapture (ラプチャー) シリーズ」を新たにラインナップに追加した．今回発表したのは80486DX (50MHz) を搭載した「Rapture55」と80486DX (33MHz) を搭載した「同33」の2機種．「Rapture55」は，S3社製チップセット仕様の高速SVGAカード，8MBのメモリ (最大32MBまで拡張可能)，240MBのSCSIハードディスク，256KBキャッシュメモリなどを標準搭載している．「同33」は，SVGAカードにET-4000チップセットを採用している．価格は698,000円 (Rapture55)，498,000円 (同33) の予定．

日本バー・ブラウン，Windows対応メータ表示ソフトを発売

日本バー・ブラウン (株) (TEL 03-3586-8141) は，MS-Windows (3.0以上) 対応のグラフィックメータソフトウェア「PCImeter」を発表した．アナログデータを収集し，収集したデータを棒グラフ，デジタルとアナログメータとしてスクリーン上に表示することができる．ハイ／ロー・アラームの設定値は，各メータの中に表示させることができる．また，各メータはハイ／ロー・アラーム値が検知されたときに出力するアラーム信号 (デジタル出力) の論理を決めることができる．DDEを介してExcelにデータを転送することができる．対応機種はIBM PC/ATと互換機，EISAバスコンピュータ，1MB以上のメモリ，PCI I/Oボードが必要．

サイリックス，486命令セット互換マイクロプロセッサを発表

米国サイリックス・コーポレーション (サイリックス (株)：TEL 045-471-1661) は，486と命令セット互換のマイクロプロセッサ (25MHz) 「Cx486SLC」を発表した．外部16ビット，内部32ビットのデータバスをもち，バスおよびピン配置が386SXと互換で，386SXを使用した設計を486レベルの性能まで容易にアップグレードすることができ，386プロセッサクラスの価格で486プロセッサの性能を実現することができるノートパソコンに適したCPUとなっている．

リンク，NetWare対応ウィルス対策用ソフトを発売

(株) リンク (TEL 03-3493-5850) は，LANのファイルサーバをベースにした世界初のコンピュータウィルス対策用ソフトウェア「LAN Protect」を発表した．これは，米国トレンドマイクロデバイス社とインテル社が共同開発した製品で，サーバへ入出力されるデータを常時自動スキャンすることによって検疫し，ウィルスの動向をユーザーの手をわずらわせることなく知らせる．Nobell NetWare 3.1X対応で，価格はサーバあたり995ドル．発売時期は，4月 (英語版)，6月 (日本語版) の予定．

Programmer's Program

3.5光磁気ディスク用デバイスドライバ

DRVR-55

次世代の記録メディアとして期待されている3.5インチ光磁気ディスク(以後3.5インチMO)は，現在128MBのタイプのものがISOによって共通規格となっている．HDDのように高速・大容量で，フロッピーディスクのようにメディアの交換が可能と，魅力ある記憶媒体である．

3.5インチMOは1991年春にIBMが販売を開始し，年末より数社から198,000円と低価格なドライブが販売されるようになって，ぐっと身近なものとなった．ただ問題は，この3.5インチMOドライブが現在自分が使用しているHDDと共存できるかという点ではないだろうか．現在使用しているのがPC-9801-27互換のハードディスクインターフェースボード（俗にSASI型インターフェースと呼ばれる）に接続されているHDDならば，まったく問題はない．ドライブと別途にメーカー指定のSCSIインターフェースボードを購入すればよい．しかし，PC-9801-55/U/L互換のSCSIインターフェースボード（以後55互換SCSIボード）に接続されるHDDだと問題は複雑である．

55互換SCSIボードに各社MOドライブを接続する

3.5インチMOドライブは，ほとんどSCSIインターフェースを採用しているが，55互換SCSIボードが高速化などを目的としてメーカー独自の機能を搭載している場合や，古いタイプのボードだったりすると，3.5インチMOドライブに対応していなかったり，扱いが異なる場合もある．3.5インチMOドライブを発売したメーカーのインターフェースボードであればROMの交換などで対応してもらえるが，HDDメーカーの中にはまだ3.5インチMOドライブを発売していないメーカーも多い．「DRVR-55」はPC-9801-55/55L/55Uや各社の55互換SCSIボードとメーカーの異なる各社の3.5インチMOドライブを接続するためのデバイスドライバである．

インストール

ディスクの中には3.5インチMO用デバイスドライバのほかにフォーマットプログラムやSCSIデバイスの確認をするためのユーティリティが含まれている．もちろん，インストールプログラムも用意されているので導入はいたって簡単である．カレントをDRVR-55のマスターディスクに移動し，MS-DOSのコマンドラインから

```
A> MOINST
```

と入力し，指示にしたがってドライブおよびディレクトリを指定していくだけである．あとはリセットするだけで，3.5インチMOドライブが利用可能となる．もちろんインストールプログラムを利用せず，自分自身で設定することも簡単である．CONFIG.SYSに登録するデバイスドライバ「MO-DRVR55.SYS」のオプション指定は2つだけである．

```
DEVICE=MODRVR55.SYS -U<n> -V<m>
```

```
3.5インチ光ディスクフォーマットプログラム Ver 1.01
                                        (c) 1992 Micro Staff
        現在のDOSヴァージョン   3.30
        ドライブ/SCSI-ID       2 (SONY SMD-C301)
        物理フォーマット       自動
        システム               転送できない
        ボリュームラベル
        媒体種別               (3.5インチ) 光ディスク

        終了

準備はよろしいですか
（はい：初期化する  いいえ：初期化しない）
      いいえ
```


ワンラインNEWS
東芝はマルチメディア機器用に映像や音声などの各種信号を多重化，分離するICを開発した．1秒間に64Kビットから1.5Mビットの通信速度に対応できる．

〈n〉には3.5インチMOドライブを接続したID，〈m〉には書き込み時にベリファイする3.5インチMOドライブのIDを指定する．MS-DOS 5.0で接続可能な3.5インチMOドライブは2台までであるが，このデバイスドライバを使えば最大7台まで同時使用が可能となる．

専用フォーマットプログラムで3.3xでも利用可能

3.5インチMOは，MS-DOS 5.0でのみサポートされている．PC-9800シリーズの場合は，MS-DOS 5.0が登場間もないこともあって，まだ普及していないのが現状である．DRVR-55は，PC-9800シリーズのMS-DOS 3.3xにおいても3.5インチの光磁気ディスクドライブを利用可能にしてくれる（MS-DOS 3.3xでは立ち上げはできない）．

日本電気のMS-DOS 5.0に付属するFORMAT.EXEでフォーマットした3.5インチMOはMS-DOS 5.0上でしか利用できないのに対し，MOFORM98.EXEでフォーマットした3.5インチMOは特殊なフォーマットにより，MS-DOS 5.0はもちろん，MS-DOS 3.3xでDRVR-98が組み込まれた環境であればデータの互換をとることができる．もちろん，このフォーマットプログラムでMS-DOS 5.0で起動できるシステムディスクを作成することも可能である（表1）．

ほかにも，接続されている各種SCSIデバイスのタイプ，メーカー，ドライブ名を調べるユーティリティ「SCSIDEV.EXE」や，挿入されている3.5インチMOの論理フォーマットを調べる「MOCHECK.EXE」などがある．

ドライブのデバイスタイプによっては接続できない

ただし，ICMのMO-3120やSONYのRMO-S350には，ドライブをHDD互換にする特殊なモードがある．このモードでは，デバイスタイプが00H(＝ハードディスク）となるため使用できなかった．マニュアルの中にも明記されているが，デバイスタイプが00Hとなる3.5インチMOドライブにDRVR-55を使うことができないのである．ドライブ購入前にあらかじめメーカーに問い合わせるなどしたほうがいいだろう．

また，この場合でもマイクロ・スタッフが発売している「MSMO-98」というSCSIインターフェースボードがあれば解決する．MSMO-98はHDDやMOドライブはもちろん，ストリーマやCD-ROMなど各種SCSIデバイスの接続を可能にする．このボードはPC-9801用以外にも，富士通FMRシリーズ・FMTOWNS用（MSMO-FM）／東芝J-3100シリーズ用（MSMO-J3）／日本IBM PS/55シリーズ用(MSMO-PS／MSMO-PS-OS2）／各社PC互換機用（MSMO-PC）が用意されている．

店頭での購入価格は，3.5インチMOドライブと180MBや240MBのHDDがほぼ同価格である．増設を考えた場合2台目のHDDか，それとも3.5インチMOドライブかと頭を悩ませるところだ．DRVR-55でボードとドライブメーカの相性を考えなくてもいいのであれば，可搬性や単位あたりのメディアの価格を考えて，間違いなく3.5インチMOドライブに軍配が上がるだろう．

■DRVR-55
販売：（株）マイクロ・スタッフ（TEL 03-3325-8128）／価格：10,000円／適応機種：日本電気 PC-9801シリーズ（ただし55互換SCSIボードとSCSIインターフェースを備えた3.5インチMOディスクが必要）／動作環境：MS-DOS 3.3以上／供給メディア：5インチ2HDおよび3.5インチ2HD同梱

表1

	MS-DOS 3.3x ＋DRVR-55	MS-DOS 5.0 ＋DRVR-55	MS-DOS 5.0 デバイスドライバなし
NEC純正FORMAT.EXEでフォーマット	×	○	◎
IBM純正FORMAT.EXEでフォーマット	×	○	○
DRVR-55付属のMOFORM98でフォーマット	○	○	◎

○：読み書き可能　◎：起動可能
注）ドライブメーカーが異なる場合，書き込みすると不都合が生じる場合もある．

オンラインソフト採集帳 18
フロッピーディスクのファイル化ツール MAHALITO

知らず知らずに机の上やキャビネットの中にフロッピーディスクは増え続けている．市販アプリケーションのバックアップやそのデータ，さらにオンラインソフトと……．この中から，必要なフロッピーディスクをすぐに取り出すことができるだろうか．またこれらフロッピーディスクのバックアップは行っているだろうか．大容量のハードディスクの普及にともない，ハードディスクのバックアップツールが数多く登場している．しかし，フロッピーディスクのバックアップはMS-DOSに付属するDISKCOPYなどで別のフロッピーディスクにコピーして，保存しているだけであろう．100枚のフロッピーディスクとそのバックアップが100枚，これらの保存場所も馬鹿にならない．

「MAHALITO」はフロッピーディスクを1枚丸ごとファイル化したり，そのファイルからフロッピーディスクを復元するツールである．またディスクコピー機能ももっている．MAHALITOによって生成されたファイルはLHAなどファイル圧縮ツールで圧縮することでフロッピーディスクのバックアップをハードディスクに効率的に行うことができる．またオリジナルのフロッピーディスクの内容次第では，2枚以上のディスクの内容を1枚にまとめることも可能である．

MAHALITOの構文

MAHALITOの構文は以下のとおり．

A> MAHALITO 〈コマンド〉［〈オプション〉］［〈転送元〉］［〈転送先〉］

コマンドにはA・E・Cがあり，省略することはできない．Aコマンドはフロッピーディスクのファイル化を行い，Eコマンドはファイルからフロッピーディスクを復元，Cコマンドはディスクコピーの指定である．

オプションはフロッピーディスクメディアの指定やシリンダ数を指定するのに使われ，-（ハイフン）または/（スラッシュ）から始まる（オプション一覧参照）．

コマンドラインパラメータの最後の2つが転送元と転送先とみなされる．Aコマンドが指定されている場合，〈転送元〉はフロッピーディスクのドライブ名，〈転送先〉には，格納するファイル名を指定する．Eコマンドが指定されている場合は〈転送元〉はデータが格納されているファイル名，〈転送先〉に復元するフロッピーディスクのドライブ名を指定する．Cコマンドが指定されている場合は〈転送元〉と〈転送先〉にフロッピーディスクのドライブ名を指定する．なおファイル名を指定する場合は拡張子は不要である．転送先がフロッピーディスクドライブの場合，前もってフロッピーディスクをフォーマットしておく必要はない．

オプション一覧

フロッピーディスクメディアタイプの指定

フロッピーディスクメディアタイプは-2HD（2HDフロッピーディスクを処理対象にする）／-2DD（2DDフロッピーディスクを処理対象にする）／-2D（2Dフロッピーディスクを処理対象にする）が指定可能で，省略した場合は-2HDオプションを選択したものとみなされる．ただしVM以前のメディアが自動判別できるフロッピーディスクドライブを持たない機種は指定に制限がある．

フォーマット情報ファイルの指定

-fオプションはフロッピーディスクのファイル化とディスクコピーのとき有効で，-fに続けてフォーマット情報ファイル（拡張子が .2HD，.2DD，.2Dのいずれかのファイル）を指定する．フロッピーディスクを読み込むときにフォーマットの解析を行わず，フォーマット情報ファイルを参照しながら処理を行う．フォーマット解析を行わないぶん処理が速くなる．フォーマット情報ファイルの指定時には拡張子の指定は不要で，上記の-2HD/2DD/2Dオプションにより，該当する拡張子が付加されたファイルが参照される．復元処理を行うときは指定しても無視される．

シリンダ数

-mオプションはAコマンドとCコマンド指定時有効で，-mに続けてシリンダ数を指定する．2HDフロッピーディスクで78シリンダ以上，2DDフロッピーディスクで81シリンダ以上，2Dフロッピーディスクで41シリンダ以上読み込みたいときに指定する．省略すると，2HDのとき77シリンダ，2DDのとき80シリンダ，2Dのとき40シリンダとして処理する．〈シリンダ数〉に指定できる数値の上限は85である．IBM PCの2HCフォーマットやFDISKで作成した512バイト／セクタのフォーマットのフロッピーディスクを処理するときはシリンダ数に80を指定すること．

ワンラインNEWS
学習研究社は情報処理技術者試験の受験者向けにビデオテープやパソコンソフト，書籍などを組み合わせたシステム教材「GIPTEC」の販売をはじめた．

復元処理を行うときや，-fオプションを指定したときは指定しても無視される．

データファイルの上書き

-oオプションはAコマンドが指定されているときのみ有効で，データファイル／フォーマット情報ファイルの同名確認をせずにファイルを書き出す．このオプションを指定しないときは，同名のファイルが存在するときオーバーライトの可否を尋ねる．

エラーがあっても処理を継続

-cオプションを指定するとフロッピーディスクでI/Oエラーが発生しても可能な限り処理を継続する．エラーのあったセクタとエラーの内容は，処理終了時にまとめて表示される．ただし，フォーマット解析の段階でI/Oエラーが発生したときは処理は継続せずに中止する．エラーがあったセクタの内容は不定なので注意して使用すること．このパラメータを指定しなかったときは，I/Oエラーが発生するとすぐに処理を中止する．

フロッピーディスクのベリファイ

-vオプションはEコマンドとCコマンド指定時のみ有効で，フロッピーディスクにデータを書き出したあとでそれが正常に読み込めるかチェックする．

フォーマットしない

-nオプションはEコマンドとCコマンド指定時のみ有効で，フロッピーディスクの物理フォーマットを行わない（無指定時は物理フォーマットを行う）．

サンプルオペレーション

フロッピーディスクの内容をファイル化する場合はAコマンドを指定する．たとえばMS-DOSのコマンドラインから

```
A> MAHALITO A B: A:¥BACKUP¥DISK001
```

と入力すれば，Bドライブの2HDフロッピーディスクを読み込み，その内容をAドライブのBACKUPディレクトリにDISK001.2HDとDISK001.DATというファイルを生成し格納する．フロッピーディスクが2DDの場合は，-2DDオプションを付加し，MS-DOSのコマンドラインから

```
A> MAHALITO A -2DD B: A:¥BACKUP¥DISK001
```

と入力する．

逆にMAHALITOで生成したファイルをCドライブのフロッピーディスクに復元する場合はEコマンドを指定する．AドライブのBACKUPディレクトリにあるDISK001.2HDとDISK001.DATからCドライブの2HDフロッピーディスクを復元する場合には，MS-DOSのコマンドラインから

```
A> MAHALITO E A:¥BACKUP¥DISK001 C:
```

と入力する．ディスクコピーをする場合はCコマンドを指定する．たとえばBドライブからCドライブに2HDのディスクコピーをする場合は

```
A> MAHALITO C B: C:
```

と指定する．

MAHALITOはMS-DOSと物理フォーマットが同じであれば，どんなフロッピーディスクもファイルにすることができる．CP/Mのディスク，汎用機で使われているIBMフォーマットのフロッピーディスクや，ワープロ専用機の文書ディスクなどもMS-DOSのファイルとして扱うことが可能になる．ただしIBM PCの3.5インチ1.44MBディスク，Macintoshのフロッピーディスク，1トラックに異なるセクタ長のセクタが存在するようなディスク，物理的なセクタ長と論理的なセクタ長が異なるディスク／セクタIDがあってセクタ本体が存在しないようなセクタがあるディスク，その他「プロテクト」が施されているディスクなどをファイル化することはできない．

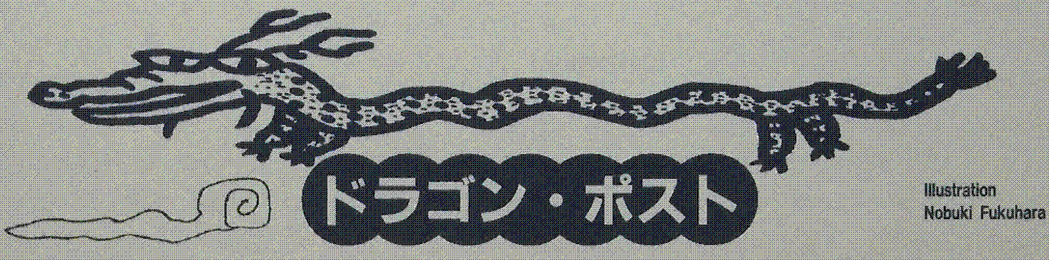
リムーバブルタイプのハードディスクや光磁気ディスクドライブも手の届く価格となっている．とくに3.5インチの光磁気ディスクドライブは1MB当たり60円と5インチフロッピーディスクと変わらない価格でメディアを入手できる．MAHALITOはこれらのメディアと組み合わせることで，本領を発揮するだろう．

■MAHALITO ver.2.10

名称：MAHALITO／分類：ユーティリティ／登録名：MHLT210.LZH／作者：ひとり上手(PC-VAN:PNJ90690)／動作環境：日本電気 PC-9801シリーズ／登録月日：1991年12月25日／復元方法：LHA E MHLT210

ファイル構成

```
Listing of archive : MHLT210.LZH

 Name           Original    Packed  Ratio   Date      Time     Attr Type  CRC
--------------  --------  --------  ------ --------- --------  ---- ----- ----
 README.DOC         1495       795  53.2% 92-04-11 02:10:02 a--w -lh5- 7BB8
 MAHALITO.EXE      30107     15152  50.3% 92-04-11 02:10:02 a--w -lh5- 8BFD
 MAHALITO.DOC      22341      7890  35.3% 92-04-11 02:10:02 a--w -lh5- 67BC
 MAHALITO.MUL       2653       922  34.8% 92-04-11 02:10:02 a--w -lh5- 87F6
--------------  --------  --------  ------ --------- --------
    4 files        56596     24759  43.7% 92-04-11 02:10:02
```



ドラゴン・ポスト

Illustration
Nobuki Fukuhara

この欄は，読者からのお便りを中心に構成します．本誌に対するご意見・ご感想などをお待ちしております．鬼より恐い読者からの声ということで，「ドラゴン・ポスト」です．本誌についているアンケートはがきにご意見の欄を設けてありますから，そこに書いてください．ペンネームでも歓迎です．ただし，ハガキには本名と住所をしっかり書いてくださいね．掲載誌をお届けします．

★QuickC for Windowsがおすすめ

QuickC for Windowsを入手し，使い始めました．これまで仕事でOS/2のプログラミングをしていた関係から，片手間のWindowsプログラミングもMS-Cを使っていたのですが，なにしろ大きなシステム(PWB派です)，遅いし，ハングの憂き目に会うことしきり，かなりツラいプログラミング環境でした．それがQC/Winは，まずWindows APであること，そして軽い，オマケにPWBと似たエディタ環境．ボクにとっては良いことづくめです．確かにWinHelpなどに難はあるけど，SDKも持っている身にはそれ程でもありません．完成してからおもむろにMS-Cでオプティマイズをかければ，かなり良好なものができます．みんなにもQC/Win，是非お勧めします!!（佐野浩之）

——本号のニュース欄でもお伝えしているように，QuickC for Windowsは，英語版が日本でもマイクロソフト（株）から発売されました．手軽なWindows処理系の登場は，Windows自体の普及にもプラスになるでしょう．

★フリーソフトを見習って！

パソコン通信を初めて一年近くたちました．そこでおどろいたのはフリーソフトウェアの質，レベルの高さにおどろいています．どれも市販のパッケージソフトにおとらない出来映えのものばかりですね．この程の物をフリーソフトウェアとして提供する人達がいるのだから，ソフトハウスの作る製品には，より一層の質・レベルの高さを求めたくなります．ソフトハウスの人もフリーソフトウェアを見習って欲しいものです．（増田雅彦）

——フリーソフトウェアの作者は，「自分の欲しいものを作る」という姿勢の人が多いようです．だから，市販ソフトと比べてもかゆいところに手の届くソフトが多いのでは．

★HDは太ったけれど……

HDDは20→40→132MBと順序良く太り，ソフトウェア(FDで供給)もたまり，スペースも問題になり始めた．最大の困り物は「カビ」．1600RPIのMT 1本あれば，200枚のFDなんか整理（つめ込み）できるんだけどなー．（あははのは）

——次第に太っていくHDD，実は価格はあまり変わらなかったりします．最近では，MOなんていう魅力的な媒体も続々と登場したりして，やっぱりパソコンは金食い虫かな？

★ゲームが増える一方

パソコン屋の人に店で使っていたTOWNSをディスプレイ付きの15万円で売ってあげるよ，と言われ，次の日に買いました．やっぱり絵のきれいなハードはいいですね．しかし，その日はのら猫に左手をかまれてゲームが出来ませんでした（家の飼い犬（3匹）が猫にかみついていた）．TOWNSでCをやろうと思ったら，16万近くかかるし（HDを入れて），通信をやろうとしたら家の電話がキャッチホンになってしまい，思うようにいかないので，ゲームが増える一方です．（松村恵一）

——犬3匹を相手にしていた猫もすごいですね．心優しい松村さんは，猫を助けようとして，逆に噛まれてしまったのでしょうか？　それにしても，ゲームの絵と音はきれいなほど楽しいのです．

★DOS/Vはいいな

WINDOWSを使うならDOS/Vがいちばん……なんですね．固定VRAMは苦しい．DOS/VだとDOS BOX内でDOSソフトが動くんですか？　NECのWIN3だとフルスクリーンしか×なんですけど，でも……PC-98用ソフトがたくさんあって，自家用マシンはやっぱりPC-98でないと，ソフトまで買い直す事になってしまうんですよね（ソフトメーカーはマシンの変更には応じてくれないんですヨ）．でも，DOS/Vはいいナー．DOS/Vマシンで98用ソフトが動くエミュレーションボードを100K程度で作る所ありませんか？　買っちゃうんだけどな！（柴田和広）

——悩みどころですよね，いままでの投資があるから……．今度発売されるワードパーフェクトはOSやマシンが変わっても対応してくれるみたいですけど，ソフトがみんなそうなるといいな．

★本にも学割が使えたら……

アセンブラの勉強をしているのですが，よく使うハード

がブラックボックス98なので，多くの本を金をたくさん使って買ったのですが，学生としては学割が利用できないのかと思っています．ソフトはボーランドなので安価なのですが，DOSは本を数冊買っただけで3万くらいは越えるし，ややこしいので大変です．さらに，やっとおぼえても，他の機種には通用しない場合が多い．ワークステーションのすごさに気を引かれつつ，しようがなく98使う日々を送っています．（佐藤和典）

——そういう人多いですよ．ほんとは使いたいマシンがあるけど，98にしてるって……．でも，ボーランドにスチューデントパックがあるのが救いかな．

★明日はアキバか図書館か

現在，某大学工学部情報工学科の発表を待っています．あと1時間位で電子メールが来るでしょう．合格していれば，明日はアキバに行ってEMSボードとMS-DOS Ver.5.0を買い，落ちていれば来年の為，図書館で勉強．うかってないかなー．（チャンドラー博士）

——このハガキは，編集部で話題を呼びました．某大学は，電子メールで合格発表をしてくれるのでしょうか．それならば，エポックメーキングなことだと思うのですが，真相をお知らせください．

★"ビギナー"になりたい

超ビギナーとして1年間過ぎましたが，まだ超ビギナー．せめてただのビギナーになりたいと思いつつ，今月もプログラマーズ・ページを読んでいます．せめて読めるところだけと思いつつ，片はしからページをめくっても読める部分がふえてこないのが悲しい．でもメゲるものか！　いつの日か立派な(???)プログラムを組んでやる!!……と夢みています．（寺尾和明）

——正しい認識だと思います．1年で自信満々だったら，あまりいいことではないと思う．プログラマーズ・ページを読んでいれば，きっといいことがあります？

★いつも上司のお古だけど……

私のパソコンはいつも上司のお古です．はじめは「これを使ってみたら？」とシャープのポータブルPC-2500S（今時お目にかかるのもめずらしい）をあてがわれ，次にNECのUV11をゆずってもらいました．今，その上司はDAをフロッピーベースで使っています．内心腹の立っている私はUV11に4MBのEMSメモリ，100MBのHDDを勝手に調達し，速度は遅くても作業環境の良いシステム構築を目ざしてがんばっています．（ドラゴンパパ）

——いいマシンは上司からという悪癖かな．有効活用を考えるのだったら，使える人から順番にが正しいと思うのだけど……．

★C++をどう思う？

Modula-2関連の記事は，そりゃもう毎回執筆者の顔が大体決まってしまう位，少ないので（日本では），「Modula-2 vs. C」のような記事には思わず期待してしまいます．とこれでCユーザはC++のことをどう思っているのでしょうか？　「あんなものCじゃねえ!!」という人がいてもいいと思うのですが……．そういう観点からぜひ「C++ vs. C」もやってほしかった（Modulaユーザから見るとC++はModula-2に毛のはえた位のことしかやってないように思えます）．（EOS.）

——EOS. さんはModula-2使いなのでしょうか．限られた執筆者の顔ぶれの一人に加わってはいかが？

★私にとってパソコンとは……

今まで私にとってのコンピュータは道具のようなものと思っていた．しかし，先日一週間ほどパソコンに触れない機会を得て，認識を改めた．別にパソコン，ワープロがなくても，レポートも原稿も何の支障もなくあげてゆけるのだ．でも，ないと何か落ち着かない．そう，私にとってコンピュータはタバコやお酒のようなものかもしれない．（タバコもお酒もダメな狂夢）

——お，鋭い指摘．パソコンを使うことが仕事でない限り，パソコンがなくても本当に困ることはないのかもしれない．

★きれいな画像が見たい

「プログラマーズ・ページ」毎号愛読しています．クイズ・ランダムスクリーンの画像も娘といっしょに動かして楽しんでいますが，使用機種がVMで，ディスプレイがデジタルなので，きれいに画像出力しないのが残念です．VMでもきれいに見えるようにはならないものでしょうか．（石田高次）

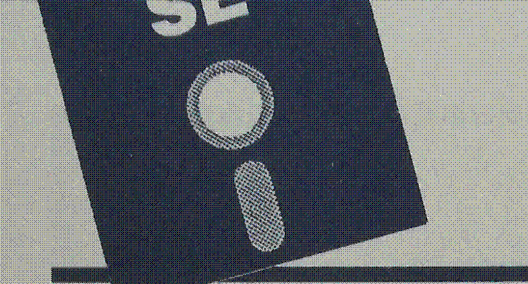
——デジタル8色表示，やはりきれいではありませんよね．でも，テレビと比べたらとても高価なパソコンなのに，16色でこの程度の画像しか表示できないなんて．理由はわかりますけどね．マルチメディアなんていってるくせに……．

★不況でヒマだというけれど

最近は不況で仕事がなくてヒマなはずなのですが，仕事をとってこなくてはいけなくて，ムナシイ忙しさです．来週からフランスまで行って仕事をとってきます．うーむ？　エイズがうつらないといいんですが……．（中島信行）

——どうしてエイズがうつるんでしょう？　コンピュータウイルスは心配ありませんか．

（敬称略）



付録ディスクのご案内

このフロッピーディスクに含まれる実行ファイル，ソースリストおよびテキストファイルは「月刊Programmer's Page」1992年６月号に掲載されたものです．

付録のフロッピーディスクは，5"2HD（８セクタ／トラックMS-DOS）でフォーマットされています．3.5"や2DDなどへのコンバートサービスは行っておりません．なお，オンラインサービスSE-NET（03-5467-0376）でも，同じ内容のファイルを用意しています．

フロッピーディスクの内容は次の通りです．本誌およびディスクに含まれるドキュメントファイルをよく読んでからご利用ください．

■ディスクのおもな内容

●付録ディスクについての説明

ディレクトリ名：ルート

収録ファイル名：

README.BAT（ディスク内容の説明を読むためのバッチファイル，PC-9801専用）

README.DOC（ディスク内容の説明，MS-DOSテキストファイル）

ONLINE.BAT（オンラインソフトの説明を読むためのバッチファイル，PC-9801専用）

ONLINE.DOC（収録されているオンラインソフトの簡単な説明）

V.EXE（ミニページャー，PC-9801専用）

BOOK.TXT（翔泳社書籍目録）

●特集「アセンブラの作り方」

ディレクトリ名：TOKU

収録ファイル名：TOKU.LZH

●実践C塾　アルゴリズム篇（p.78）

ツリー・その２

ディレクトリ名：JUKU

収録ファイル名：JUKU.LZH

●イカすBASICプログラミング入門（p.88）

マイナーチェンジ編

ディレクトリ名：BASIC

収録ファイル名：BASIC.LZH

●98活用アセンブラ技法（p.99）

RS-232C編

RS-232Cデバイスドライバ

ディレクトリ名：98ASM

収録ファイル名：98ASM.LZH

●PRODUCTS REVIEW（p.113）

Microsoft QuickC for Windows

ディレクトリ名：REVIEW

収録ファイル名：QCWIN.LST

●綴じ込み付録

ディレクトリ名：DATA

収録ファイル名：8086.TXT（MS-DOSテキストファイル）

●オンラインソフト採集帳（p.140）

今月号で紹介したフロッピーディスクファイル化ツール「MAHALITO」と，１つまたは複数のファイルの圧縮および「.LZH」という拡張子をもった圧縮ファイルの復元をする高圧縮復元書庫管理ソフト「LHA」が収録されています．MAHALITOはLHAによって，LHAは自己復元型で，それぞれ圧縮されています．復元方法についてはONLINE.DOCを参照してください．

ディレクトリ名：ONLINE

収録ファイル名：

MHLT210.LZH（MAHALITO v.2.10，PC-9801用）

LHA213.EXE（LHA v.2.13，MS-DOS汎用）

●クイズ・ランダムスクリーン（p.146）

クイズ解読ツール「PUZLE」

本誌掲載のクイズ・ランダムスクリーンの画像をディスプレイ上に再現し，マウスを使って並べ換えることにより，解答を得ることができるユーティリティです．PC-9801シリーズ専用です，詳しい使い方はPUZLE.DOCを参照してください．

ディレクトリ名：OMAKE

収録ファイル名：

PUZLE.EXE（解読ツール，PC-9801専用）

PUZLE.DOC（PUZLEの説明）

CONFIG.SYS（PUZLE用CONFIG.SYS）

AUTOEXEC.BAT（PUZLE用AUTOEXEC.BAT）

RNDSCREN.LZH（画像データ）

NOZOMI.LZH（５月号の解答データ）

NOZOMI.BMP（Windows 3.0の壁紙）

※使用上の注意

本誌に掲載されたプログラムおよびデータを利用するには，MS-DOSの基礎知識が必要です．このディスクには，MS-DOSのシステムは入っていません．別途，MS-DOSが必要となります．また，ソースファイルのコンパイルなど，別途コンパイラやユーティリティが必要な場合があります．

■圧縮ファイルについて

ディスクスペースを有効に使うために，多くのファイルは圧縮した上で収録しています．「.LZH」という拡張子のついたファイルは，高圧縮書庫管理ユーティリティ「LHA」によって圧縮されたファイルです．これらのファイルを利用するには，まずLHAで復元する必要があります．LHAの利用方法については，ONLINE.DOCをご覧ください．

バーチャルスペースガイド

P子　某ソフトハウスで企画営業を担当している．26歳．
C平　某メーカーのシステムソフトウェア開発部門に所属．25歳．

C平　もー，マルチメディア，マルチメディアって何なんだろう．音に映像に時間概念……，ソフトを書くのも大変になるばっかりだよ．

P子　けっこうまいってるじゃん．やっぱり，マルチメディアがらみの仕事が入ってるの？

C平　あれだけ，鳴りもの入りでやってるんだから，どのメーカーも何かをしなければならないって状況なわけ．でも，Windowsが3.1になってAPIがうんと増えたっていうように，いろいろなメディアのことを考えてソフトを作るのは大変だよね．

P子　このごろは，マルチメディアだけじゃなくて，バーチャルリアリティってのもあるけど．

C平　やりすぎなんだよね．研究所でディスプレイかぶって実験しているのならいいんだけど，ビジネスショーなんかでバーチャルリアリティなんてやられちゃうとね．やっぱり，いまのところは映画の中だけにして欲しいね．

P子　「ブレインストーム」とか，「トータルリコール」とか？

C平　現実と夢の区別がつかなくなるという手法は，小説や映画などで昔からあったよね．とくに「トータルリコール」のように映像で実世界と虚の世界を見せられたら，見ているほうには区別がつかないのは当然だけど，わからなくなってしまうこと自体があの映画のおもしろさになっている．

P子　だよね．小説も映画もどういう情報を与えるか，作者が制限しているわけだから，「藪の中」みたいなおもしろさってあるものね．

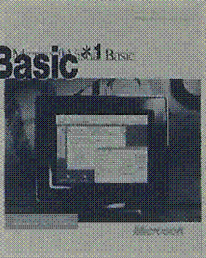
■トータルリコール

（1989 年・米）
監督：ポール・バーホーベン
出演：アーノルド・シュワルツェネッガー
4,700 円（税別）　パイオニアLDC

火星に行ったことがないのに，火星の夢にうなされる男が，火星へのバーチャルトリップを試みる．バーチャルトリップに失敗した男が家に帰ると，知人や妻から命を狙われる．実は，火星にいた頃の記憶を消されて地球で生活していた？運命に導かれるままに本当に（？）火星に出発する男．そこに展開されるのは，実世界，それとも…….

読者プレゼント

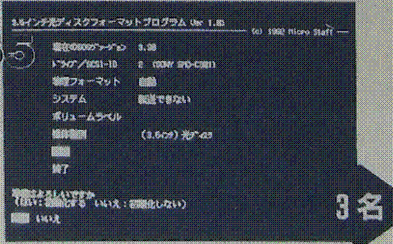
1 Microsoft Visual Basic*1



提供：マイクロソフト㈱

2名

2 DRVR-5
(PC-9801対応)



提供：(株)マイクロ・スタッフ

3名

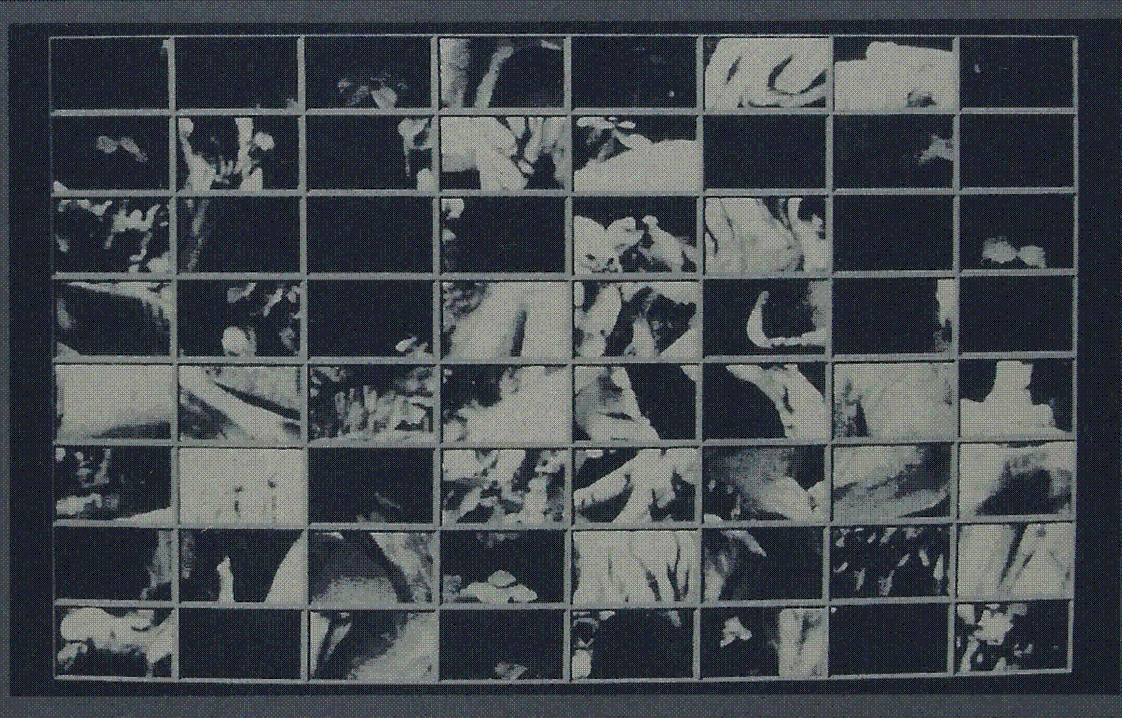
応募は本誌に添付のアンケートはがきをご利用ください．(1992年 6 月17日消印有効)．なお，応募者多数の場合は，抽選になります．当選者発表は発送をもって代えさせていただきます．

＊1：動作環境については「NEWS & NEW PRODUCTS」（134 ページ）を参照してください．

クイズ・ランダムスクリーン

ある１枚の画像を64分割し，ランダムに並べ替えてあります．
さて，もとの画像は何を写したものでしょうか．

ヒント：酔っぱらい



正解者の中から30名様に本誌表紙イラスト（日暮修一画）入り月刊プログラマーズ・ページ特製オリジナルマグカップをさしあげます．

応募方法：本誌に添付のアンケートはがきをご利用ください．

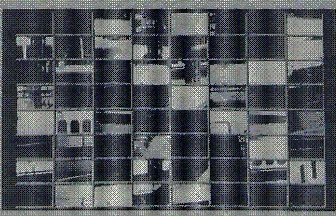
締め切り：1992年６月17日（消印有効）

なお，当選者の発表は発送をもって代えさせていただきます．

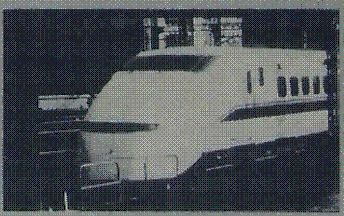
＊画像データは，付録ディスクやBBSでもサービスしています．

５月号の問題と正解

【問題】



【正解：のぞみ】



SE-NETからのお知らせ

SE-NETでは開設以来，オンラインサービスを行ってきましたが，回線混雑のため非常に接続しにくい等の問題が出てまいりましたので，GUESTによるダウンロードは中止させていただきます．今後は読者のみなさまと編集部とのコミュニケーションの場としてご利用ください．

●BBSへの接続

SE-NET　03-5467-0376

通信速度	300/1200/2400bps
接続プロトコル	CCITT V.22bis/V.22/V.21
エラー訂正プロトコル	CCITT V.42, MNPクラス5
漢字コード	シフトJIS

＊移転のため電話番号が変わっておりますので，ご注意ください．

月刊プログラマーズ・ページ バックナンバーのご案内

Programmer's Page MONTHLY MAGAZINE FOR PROGRAMMERS

■定価・各980円

1991年1月号「特集・プログラマの仕事場」・2月号「特集・Windowsの時代である！」・3月号「特集・この言語を使いなさい!!」・4月号「特集・C読本」・5月号「基礎力としてのアセンブラ」は品切れです．

1991年6月号 5インチFD付き

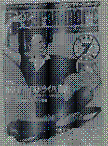
特集・研究「プログラミング統合環境」

今流行のエディタ，コンパイラ，リンカ，デバッガを統合した開発環境について，その歴史的系譜から最新のシステムまで徹底的に検証する．

1991年7月号 5インチFD付き

特集・最新「デバイスドライバ」研究

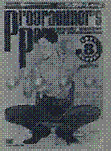
パソコンの拡張性を支えるソフトウェアインターフェースであるデバイスドライバ作成の技法と特徴をMS-DOS, Windows, OS/2の各環境ごとに解説する．



1991年8月号 5インチFD付き

特集・はじめてのオブジェクト指向プログラミング

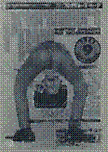
"オブジェクト指向"の歴史的な発展の過程から基礎概念，分析・設計の方法までを網羅的に解説する．



1991年9月号 5インチFD付き

特集・グラフィックスプログラミングの世界

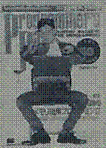
基礎からフレームバッファの活用，レイトレーシング，Windows，IBM PCのグラフィックスなど，その技法を徹底的に解説する．



1991年10月号 5インチFD付き

特集・286/386メモリ活用プログラミング

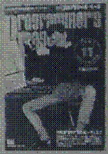
バンク式メモリ，EMS，XMSからプロテクトモードまで，640KBの壁を破るための数多くのプログラミングテクニックを解説する．



1991年11月号 5インチFD付き

特集・Windowsプログラムの作り方

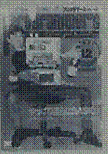
正統派MS-C＋SDKをはじめ，VisualBasic，BC＋＋，TPWなどの新しい開発環境，ツールによるWindowsプログラムの作り方を探る．



1991年12月号 5インチFD付き

特集・やっぱり僕らはBASIC／

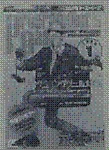
構造化プログラミング，マルチランゲージに対応した強力なプログラミング言語として生まれ変わった現在の進んだBASIC，MS-BASIC 7.1，TrueBASICからVisualBasicまでを紹介する．



1992年1月号 5インチFD付き

特集・プログラミングツール大集合

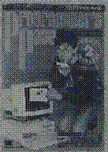
プログラム開発，メンテナンスに欠くことのできないプログラミングツール．その利用法から，使いこなしのコツまでを紹介する．



1992年2月号 5インチFD付き

特集・通信ソフトの作り方

RS-232Cを使用した通信の概要を解説し，ライブラリ，ライブラリを用いた通信用マクロ言語を作成，BBSなどにアクセス可能な通信プログラムを完成する．



1992年3月号 5インチFD付き

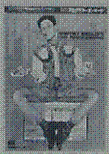
特集・Cじゃなくてもいいじゃない

C言語の実体をとらえなおし，PascalやMODULA-2など，ほかのプログラミング言語と比較することでプログラミング言語の能力と性能を解説する．

1992年4月号 5インチFD付き

特集・新しいDOS環境を探検する

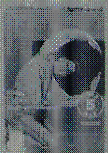
これまでのDOS環境に変化をもたらしたMS-DOS5.0，DOS/V上でのプログラミングを探る．バージョンアップしたMASM6.0もあわせて紹介．



1992年5月号 5インチFD付き

特集・Cの学び方・教え方

もっとも多く使用されていながら，学ぶのが難しいといわれるC言語の学び方から教え方まで，プログラミング学習のよりよい方法を模索する．



「月刊プログラマーズ・ページ」のバックナンバーは，ご注文いただければ，全国どこの書店でもお求めになれます(書店に届くまで1か月ほどかかります)．小社に直接ご注文なさるときは，何月号かを明記の上，現金書留にて雑誌価格＋送料を右記までお送りください．送料は1部81円です（ただし7月号は1部86円)．2冊以上の場合は，号数・冊数により送料が異なりますので，小社までお問い合わせください．

〒106 東京都港区西麻布2-22-1
スリーアップ西麻布ビル
(株) 翔泳社
営業部PP係宛
TEL 03-5467-0361

Programmer's Page

MONTHLY MAGAZINE FOR PROGRAMMERS

月刊**プログラマーズ・ページ7月号**

(6月18日発売)

付録: 5″ 2HDフロッピーディスク付き!!

特集

過激なプログラミング入門

ハードウェアから入るか?問題解決から入るか?

プログラミングというものをどうとらえるかによって，その学習方法は大きく変わる．コンピュータの内部構造を理解した上ですべきだと考えるか，問題解決のみに集中して行うか？ 2つの対局にある考え方から，やや過激ともいえるアプローチによってプログラミング入門の道筋を探る．

月刊プログラマーズ・ページ　1992年6月号
1992年6月1日発行
発行所 株式会社翔泳社
東京都港区西麻布2-11-1スリーアップ西麻布ビル
TEL 03-5467-0361　〒106
発行人 長廻健太郎
編集長 清水　隆
副編集長 鈴木嘉平
編集スタッフ 杉野貴美子＋芳山充弘
表紙デザイン SE（翔泳社デザイン研究所）
広告スタッフ 相澤正夫＋澤本良知＋種村嘉彦
協力 根本裕三郎＋(株)ドモン・マインズ＋小林弘人(図版デザイン)
写植 株式会社東巧社＋株式会社トーヨー・プリント社
印刷 凸版印刷株式会社
定価 980円（本体951円，税29円）

© 1992 SHOEISHA
本書の一部または全部を著作権法の定める範囲をこえ，無断で複写，転載，テープ化，データファイル化することを禁じます．
＊本書の内容に関するご質問は，ご面倒でも必ず文書にて編集部までお問い合わせください．
＊本誌に記載した会社名，ハードウェア・ソフトウェアの商品名は，各社の商標または登録商標です．

EDITOR'S PAGE

●宇宙の始まり，そしてそのゆく末を考えるのはたのしいものだ．ビッグバンで始まった宇宙は，再び収縮に転じるのか，それとも膨張を続け気の遠くなるような時間の果てに陽子が崩壊しすべてが無に帰するのか，など興味がつきない．ところが，「アインシュタインとマリリン」というイギリスの映画があって，その中のアインシュタインの言葉にドキっとさせられたことがあった．映画の中で相対性理論のファンであるマリリンは，おもちゃを使ってドップラー効果など相対性理論への理解を示してみせるのだが，アインシュタインは「知っていることと真実とは違う」というのである．真実は，常人には理解できない方程式の中にあり，その結果だけを知識として得ていることとはまったく別なのだと説くのである．映画はそうした問題提起から，核の問題へつなげていったという記憶があるが，脳天気な宇宙論ファンとしては身につまされる言葉ではあった．(TA)

●突然，ボーランド社のデベロッパーズコンファレンスにプレスとして参加することになった．アメリカに行けるのは嬉しいのだが，体は風邪をひきこんでヘロヘロである．うまくするとフィリップ・カーン氏のサックスの生演奏が聞けるかもしれないので，なんとかがんばらねば．向こうで面倒をみてくれるのは，日本語が達者で金髪・身長2メートルの大男だとか……，生きて帰ってこれるかなぁ？（嘉平）

●突然，副編集長がアメリカへ行ってしまい，しかも恐怖のゴールデン・ウィーク進行（とうぜんGWは印刷所もお休みになるのでそのぶん早く雑誌を作らなければいけない）で，いつもならだいたい一段落したところで，この「EDITOR'S PAGE」を書いているのですが，実はまだパニックの真っ最中です．たくさんお休みするためには，そのぶん働かなければならない，なんとモットモなことか．今年のゴールデン・ウィークは特別に予定もないし，ゴールデン・ウィークなんてなくてもいいな，と，思っているのは私だけでしょうね，きっと．(ま)

●なんとなく今年の春はちょっと変だ．毎週末には雨が降るし，2日とまともな天気が続かない．このまま黄金週間に突入なんて悲しすぎる．(yoshi)

●お詫び

プログラム「遊」工房は，著者の都合により休載いたしました．

愛読者カードご記入のお願い

本誌をお買い上げいただき、誠にありがとうございます。
お手数ですが、今後の誌面づくりの参考にさせていただきたいと存じますので、
各欄にご記入の上、ご返送くださるようお願い申しあげます。

月刊プログラマーズ・ページ READER'S SURVEY CARD 愛読者カード

本誌の号数　　年　　月号

本誌を知った理由
□1. 書店店頭　□2.他社出版物(　　　)
□3.小社出版物　□4. DM　□5.口コミ

本誌を購入した書店　　市(区)　　書店

パソコンの所有形態(機種名)
□1. 個人所有　(　　　)
□2. 職場にある(　　　)
□3. 持っていない

よく使用するソフトウェア

よく使用するOS

今月号でおもしろかった記事(いくつでも)

今後取り上げてもらいたい特集のテーマ

貴方が選ぶベスト3

パソコン雑誌	①	②	③
ソフトウェア	①	②	③
パソコン本体	①	②	③

本誌についてのご感想、お気づきの点、その他どのようなことでも結構ですので、ご意見をお聞かせください(「ドラゴン・ポスト=読者の声」欄に掲載させていただく場合もあります。ペンネームをご希望の方はあわせてご記入ください)。

月刊プログラマーズ・ページ 広告資料請求カード

8	会社(学校)名		9	所属(学部)		10	従業員数	

※下記の該当する番号をご記入ください。

11	業種		12	職種		13	パソコン利用状況	

業種　①製造業　②金融・保険・不動産業　③商社・卸・小売業
④建設業　⑤通信・運輸　⑥システム・ソフト
⑦その他サービス業　⑧公務員　⑨その他

職種　①コンピュータ(ハード)技術者　②コンピュータ(ソフト)技術者　③エレクトロニクス技術者
④プログラマ　⑤その他の技術者　⑥専門職
⑦事務・営業職　⑧経営管理職　⑨商工サービス
⑩教職　⑪学生　⑫その他

パソコン利用状況　①会社で利用　②個人で利用　③会社でも個人でも利用　④利用していない

●希望される広告資料　　年　　月号

広告資料請求No. / 希望項目												
A 購入希望	A	A	A	A	A	A	A	A	A	A	A	A
B 説明希望	B	B	B	B	B	B	B	B	B	B	B	B
C 資料希望	C	C	C	C	C	C	C	C	C	C	C	C

定期購読のご案内

綴じ込みの振替用紙を使って、本誌の定期購読をお申し込みになれます。
1年間12冊＝11,760円（本体11,417円/税343円）
郵送料サービス

郵便はがき

料金受取人払
麻布局承認
905
差出有効期間
平成６年４月
30日まで
切手不要

106-00

東京都港区西麻布2-22-1
スリーアップ西麻布ビル
株式会社 翔泳社
マーケティング部 行

月刊プログラマーズ・ページ READER'S SURVEY CARD 愛読者カード

氏名	性別 □1男 □2女	生年月日 19　年　月　日
住所 〒		電話 （　　）　―

職業
□1ソフト技術者 □2ハード技術者 □3技術系会社員
□4事務系会社員 □5営業系会社員 □6教師
□7一般公務員 □8経営管理職 □9理系学生
□10文系学生 □11その他（　　　）

ユーザー区分
□1プロ
□2マニア
□3一般ユーザー

プレゼント記入欄	クイズ・ランダムスクリーン回答欄
希望商品番号	答

郵便はがき

料金受取人払
麻布局承認
906
差出有効期間
平成６年４月
30日まで
切手不要

106-00

東京都港区西麻布2-22-1
スリーアップ西麻布ビル
株式会社 翔泳社
月刊プログラマーズ・ページ
広告資料請求係行

＊当社で記入

1	SE	2	受付No.	

月刊プログラマーズ・ページ広告資料請求カード

3	会員番号	

＊すでに会員番号をお持ちの方はご記入ください。

4	フリガナ 氏名	姓　　名	5	生年月日	19　年　月　日
6	自宅住所	〒　電話（　　）			
7	会社住所	〒　電話（　　）			

○資料送付希望先の番号に印をつけてください。

払込票

通常払込料金加入者負担

口座番号	東京 4	154493
加入者名	株式会社 翔泳社	
金額	11760	
払込人住所氏名	※	
受付局日附印		
備考		

切り取らないで郵便局にお出しください。

記載事項を訂正した場合は、その箇所に訂正印を押してください。

払込通知票

通常払込料金加入者負担

口座番号	東京 4	154493
金額	11760	
料金		
払込み		
特殊	円	
備考		
受付局日附印		
加入者名	株式会社 翔泳社	
払込人住所氏名	※(郵便番号　　) TEL(　　)	

各票の※印欄は、払込人において記載してください。

この払込通知票は、機械で使用しますので、下部の欄を汚さないよう特に御注意ください。また、本票を折り曲げたりしないでください。(郵政省)

株式会社

米国でのボーランド社に
合併劇。日本法人はどのよ
っていたが、今年1月1日
ンがアシュトンテイト(株)
新たなスタートを切った。
ータベース分野でも業界を
今後どのような展開を見
谷区の真新しい本社オフ
原幸吾氏を訪ね、新生ボ
うかがった。

ス、表計算、言語という3つの顔を持っていました。そしてそれぞれが同じ程度の規模でしたね。それが今回の合併により、データベースを中心とした会社であるということが明確に打ち出されたことになります。事実、売上全体の5割以上をデータベースが占めるということになっていますし。

したがって今後、日本の市場でデータベースをどのように啓蒙・普及させていくかということは、われわれの大きなテーマです。

では、この特殊性を抱えた日本市場で、ボーランドはリレーショナルデータベースの世界をどのように広げていこうとしているのだろうか。

「データベースに関しては、日本のユーザーは間違った洗脳をされてしまっているんじゃないかと思うんですよ、他の国の状況と比較して。日本ではカード型データベースという言い方がよくされています。カード型というユーザーインターフェース自体はわれわれ

ADOXでも持っていますし、るものではないんですが、そこり目がいって、データベースに基本的に重要な事柄についてのあいまいにされている。

ネスに使う場合、使いやすいとはもちろん大切な要素ですが、企業にとって一番貴重な財産のはデータそのものなんです。

そのデータを入力したり、まテナンスしたりするコストとい一番大きい。したがって、いったデータを将来にわたって活づけられることがデータベースて最も重要です。そこから〝標というものの意味と価値が出てだろうと思います。ユーザーのには企業内で上から下までソリューションを提供できるなようデータベースを選んでいただきたいと、これから大いに言っていくつもりです。

データベースを会社の中で使うということは、つまりシステムを構築するという大きな意味を持っているわけで、そういう観点からLANの必要性も出てくるんだと思います。データベースを構築し、きちっとビジネス上でメリットを出せるような形でLANを導入しなければ、LANをいったい何に使うの、という話になってしまいます。

定期購読のご案内

■定期購読は、1年間12冊と限定させていただきます。

■購読料のお支払は、この振替用紙をご利用ください。銀行振込、現金書留によるご送金は、ご遠慮ください。

■振替用紙に、ご住所、お名前をご記入のうえ、郵便局の貯金窓口にてお支払ください。

■小社より領収書の発行はいたしませんので、郵便局にて「郵便振替払込金受領証」をお受取ください。

■お申し込みいただいた方には、弊社より直接発送いたしますので、到着が発売日より遅れることがございますが、ご了承ください。

月刊プログラマーズ・ページ　定期購読直接申し込み書

この欄は、加入者あての通信にお使いください。

Programmer's Page MONTHLY MAGAZINE FOR PROGRAMMERS

年　　月号より

年間購読料 11,760 円(税・送料込み)

■ご希望の購読開始月号をご記入ください。

■送付先（住所・氏名は正確にご記入ください）

住所	□□□-□□	フリガナ	
		氏名	
		TEL	

この払込通知票は、機械で使用しますので、下部の欄を汚さないよう特に御注意ください。また、本票を折り曲げたりしないでください。　（郵　政　省）

郵便はがき

106-00

東京都港区西麻布2-22-1
スリーアップ西麻布ビル

株式会社 翔泳社

月刊プログラマーズ・ページ
広告資料請求係行

2	受付No.	

マーズ・ページ広告資料請求カード

＊すでに会員番号をお持ちの方はご記入ください。

名	5	生年月日	19　年　月　日

□10文系学生　□11その他（　　　）　ユーザー

プレゼント記入欄	クイズ・ランダムスクリーン回答欄
希望商品番号	答

6	自宅住所	〒　電話（　　）
7	会社住所	〒　電話（　　）

資料送付希望先の番号に○印をつけてください。

綴じ込みの

Programmer's Page MONTHLY MAGAZINE FOR PROGRAMMERS

広告企画

ちょっと気になるこの会社

ボーランド株式会社

昨年後半の大きな話題となった米国でのボーランド社によるアシュトンテイト社の吸収合併劇。日本法人はどのような形に落ち着くのか注目を集めていたが、今年1月1日をもって(株)ボーランドジャパンがアシュトンテイト(株)と合併し、ボーランド(株)として新たなスタートを切った。

言語分野での実績に加え、データベース分野でも業界をリードする地位を占めた同社は、今後どのような展開を見せてくれるのだろうか。東京・渋谷区の真新しい本社オフィスに、マーケティング部長の篠原幸吾氏を訪ね、新生ボーランドの今後を中心にお話をうかがった。

データベース分野中心に変貌したボーランド

BORLAND C++やTURBO C++など言語ソフトでは読者にすっかりおなじみのボーランドが、今回のアシュトンテイトとの合併により、データベース分野のナンバーワン企業へと一気に踊り出た。次代データベース・ソフトの標準を狙う〝PARADOX〞と、全世界に400万ユーザーを数えるRDBの世界標準〝dBASE〞の2大ソフトを擁して、ボーランドは日本のデータベース市場でどんな旋風を巻き起こそうとしているのか。マーケティング部長の篠原氏は、合併後のボーランドについて、こう語る。

「ボーランドはこれまで、データベース、表計算、言語という3つの顔を持っていました。そしてそれぞれが同じ程度の規模でしたね。それが今回の合併により、データベースを中心とした会社であるということが明確に打ち出されたことになります。事実、売上全体の5割以上をデータベースが占めるということになっていますし。

したがって今後、日本の市場でデータベースをどのように啓蒙・普及させていくかということは、われわれの大きなテーマです。

現在、パソコンベースのデータベースでは、われわれが全世界で7割ぐらいのシェアを占めています。ところが日本では、いわゆるカード型データベースを含めたデータベース市場全体でみると、そこまでの数字にはなっていません。日本と米国、あるいは他の諸国を比べますと、日本はデータベースに関しては特殊な市場であると言えます。何が特殊かというと、日本ではリレーショナルデータベースの市場よりも大きいぐらいに、カード型データベースと呼ばれるフラット型データベースの市場が存在しています。そこが欧米などと比較したときの大きな違いでしょうね。」

リレーショナルDBの啓蒙と普及に本腰

では、この特殊性を抱えた日本市場で、ボーランドはリレーショナルデータベースの世界をどのように広げていこうとしているのだろうか。

「データベースに関しては、日本のユーザーは間違った洗脳をされてしまっているんじゃないかと思うんですよ、他の国の状況と比較して。日本ではカード型データベースという言い方がよくされています。カード型というユーザーインターフェース自体はわれわれのPARADOXでも持っていますし、否定するものではないんですが、そこにばかり目がいって、データベースにとって基本的に重要な事柄についての認識があいまいにされている。

ビジネスに使う場合、使いやすいということはもちろん大切な要素ですが、基本的に企業にとって一番貴重な財産というのはデータそのものなんです。そして、そのデータを入力したり、またメンテナンスしたりするコストというのが一番大きい。したがって、いったん入れたデータを将来にわたって活用しつづけられることがデータベースにとって最も重要です。そこから〝標準品〞というものの意味と価値が出てくるんだろうと思います。ユーザーの皆さんには企業内で上から下までソリューションを提供できるなようデータベースを選んでいただきたいと、これから大いに言っていくつもりです。

データベースを会社の中で使うということは、つまりシステムを構築するという大きな意味を持っているわけで、そういう観点からLANの必要性も出てくるんだと思います。データベースを構築し、きちっとビジネス上でメリットを出せるような形でLANを導入しなければ、LANをいったい何に使うの、という話になってしまいます。

したがって、今後、データベースに関する啓蒙に関しては、積極的に取り組んでいこうと考えています。」

UNIX環境への ソリューションも展開

最近、ビジネスの世界では、LANやUNIXがクローズアップされるようになってきた。これらの動きに対するボーランドの戦略について、次にうかがってみよう。

「もちろん、われわれの製品戦略は、もっと上のほうまでソリューションを広げていく考えです。具体的にはdBASEの場合ですと、現在、UNIXバージョンを開発中です。これはもうまもなく日本語版もリリースします。

パソコン上で運用していただいているユーザーの中で、かなりパフォーマンスが必要になってきた、非常にたくさんのデータエントリーをしなければいけない、という状況を抱えているユーザーへの提供です。

こうした問題は、ネットワーク上でやるということがひとつの解決ですが、それでも限界になってきて、もっと処理速度をあげたいという場合には、UNIXは非常にいい解決法ですね。UNIXではマシンがたいへん高速化しますし、UNIX上のdBASEというものを考えますと、UNIXのサーバーが1台あって、それに対するアクセスというのはVT100のターミナルを使った単純な形ですが、パフォーマンスはよくなるんですよ、パソコンネットワークと比べますと。値段的にも端末がインテリジェントでなくても使えますから、ネットウェアを使って全体を構築するよりもかえって安い場合も十分有り得るんですね。そういう形でスケールアップできるソリューションを提供しましょうということです。

さらにその上に、インターベースというデータベースエンジン、サーバーエンジンですね、これを持っています。現在、その日本語バージョンを開発していこうとしています。

ボーランドのデータベース戦略は、今後インターベースが中核になって、UNIXワークステーション、その上のVAXやVMS環境まで対応し、バックエンドのサーバーとして動くエンジンの提供をしていくということになります。また、そこでは今までDOSやWindows上の製品がすべてそこにアクセスできるようにしていこうという考えで、既存のdBASEやPARADOXのユーザーがスムーズに処理能力をアップしていけるようにソリューションを提供するというのが、今後の製品戦略ということになります。」

表計算、言語分野でも 新製品が続々と

データベースを核としたソリューションの提供を着々と進めるボーランドだが、その他の分野、表計算そして言語関係ソフトの製品展開も目が話せない。

「日本ではこれまで、ボーランドというと言語の会社だということでしたね。ここに今回RARADOXとdBASEを持つことでデータベース分野でも進出を果たした。次は、これに表計算分野を加える考えです。ボーランドはQUATTROという表計算ソフトを持っていますので、これの日本語バージョンをできるだけ早く投入しようと考えています。

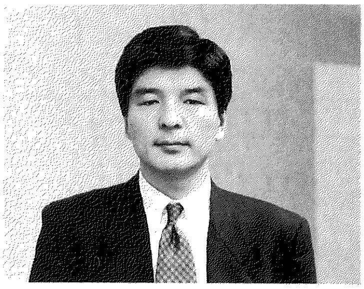
インターオペラビリティという言葉を使っていますが、インターベースエンジンを核にして、データベースだけではなく、言語、表計算、あらゆるソフトを共同で利用できるような形にしていくというのがわれわれの提唱です。製品群が全部一体化して動くというようになるんですね。そういう環境を米国ではすでに作っていますから、それを日本でも早く実現していきたいと思っています。また、そういう環境を日本で提供するために開発効率をいままで以上に高めて、開発陣も強化しています。

QUATTROは最初にQUATTRO Pro for Windowsという Windows版を日本では提供する予定です。秋までには出したいですね。」

さて、読者には身近な言語ソフトではどんな新しい動きがあるのだろうか。最後に言語分野の今後の予定を教えていただいた。

「現在BORLAND C++の3.0の発売が一番新しい話題ですが、それに引き続いてTURBO C++ for Windowsが出ます。これが今年の目玉ということになりますね。C++ 3.0では、機能的にこれ以上ないというぐらいに仕上がっています。これまでユーザーから要望の高かった最適化の機能向上をはじめ、コンパイルの速度向上など非常に盛り沢山の機能を持たせています。

それと、われわれのひとつのネックであった、複数のプラットフォームへの対応ですが、これも今後は、98版以外に開発環境として他のプラットフォームもサポートしようということで動いています。まだ時期等は申し上げられない段階ですが、期待していていただきたいと思います。」



マーケティング部部長 篠原幸吾氏

高水準の技術開発にはパーフェクト精神が根幹です。

ワンランク上の提案型ソフトウェアハウス＝エルテックスは、ユーザーや上司の求めている以上の開発を追求していく、我がままプロ集団の組織。常に高レベルの開発を行っていくためには、たえず外部の開発にアンテナを張りめぐらしておかなくてはなりません。その一環として、毎年数名、ラスベガスのコムデックスショー等、１週間程度の視察旅行に参加します。パーフェクトなものを求め続けていく技術者たちに、私たちは、たえず評価を与えていきます。

技術者随時募集

会社概要

ELTEX

設　立	昭和60年12月14日
資本金	1000万円 ⑴ネットワークの端末システムの開発 ⑵ＣＤ－ＲＯＭ、ＬＤなど新しいメディアに対応したシステムの開発 ⑶ＯＣＤ、ＣＡＤなどのソフトウェアプロダクトの開発、販売 ⑷ビジネスパッケージの開発、販売
特　徴	大手クライアント向けにネットワークやマルチメディアなどのシステム開発を幅広く対応している。また、手書文字読み取りシステム「読造」*1、水道工事店向け「配管ＣＡＤ」*1などの自社ソフトウェア製品の開発販売や、電子回路ＣＡＤ「Ｔａｎｇｏ」シリーズ*2の日本語化など積極的な事業展開をしている。 *1　当社で開発し三菱系ディラーで販売しているプロダクト *2　米アクセルテクノロジ社の製品をＰＣ9801への移植と日本語化したプロダクト、㈱ソーテック工人舎事業部で販売している。

募集要項

職　種	①システムエンジニア　②プログラマー
資　格	専門卒以上20〜36歳の方　※Ｃ言語・アセンブラ言語経験者大歓迎　※未経験者でも好奇心・向学心の旺盛な方は歓迎します。
給　与	17万円（専門卒初任給）〜46万円（36歳課長、諸手当含む） 24歳例：23万5000円（情報処理1種手当・住宅手当含む） 30歳例：31万6000円（中級SE、情報処理1種・住宅手当含む）
手　当	時間外手当、役職手当（5000円〜６万円)、住宅手当（１万5000円)、資格手当（２種：5000〜特種：２万円)、通勤手当
賞　与	賞与年２回（平成１年度実績年間５ヵ月)、決算賞与年１回（平成１年度実績１ヵ月)
勤務地	本社（横浜駅近く） ◇派遣・出向はありません。
勤務時間	フレックスタイム制（コアタイム10:00〜16:00）試用期間中は標準時間9:00〜17:45
休日休暇	完全週休２日制（土・日)、祝日、年３回連休（GW・夏期・年末年始)、有給・特別・休暇
福利厚生	各種社会保険完備、退職金制度、財形貯蓄、社員旅行、海外研修、定期健康診断、社員寮、スポーツクラブ法人会員
応募方法	電話連絡の上、履歴書(写真貼付)、職務経歴書を持参し来社ください。郵送も受け付けます。
連絡先	〒220 横浜市西区南幸二丁目20番５号　東伸24ビル５階　℡(045)322-9667　総務課

採用HOT・LINE

TEL.045-322-9667

株式会社エルテックス

ADO TERADRIVE ゲーム作家新人賞コンテスト

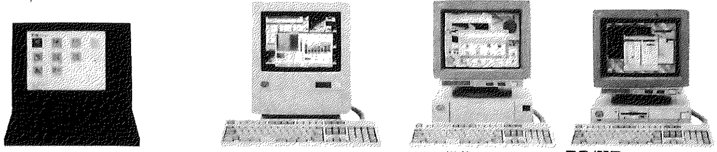
TERADRIVEを使ったゲームコンテスト どしどし作ってご応募下さい。
★受付はT・ZONE各店にて

主催：(株)亜土電子工業
期間：平成4年4月15日～平成4年7月31日迄
発表：平成4年8月31日

《賞品》 ゲーム作家新人賞
1位 アメリカ西海岸ツアー旅行（副賞50万円）
2位 ハワイ旅行（副賞30万円）
3位 香港旅行（副賞20万円）
4位 TERADRIVE
5位 TERADRIVE特別割引券

IBM Net Ware 取扱開始！ NEWマシンぞくぞく大量入荷!!

PS/55 note　PS 55Z 30U　PS 55Z 10S　PS 55Z 10T



大容量80MBハードディスク内蔵 S18 SV8 S28 SA8
パワフルなDOS/Vノート
80386SX(16MHz)2MB FDD内蔵

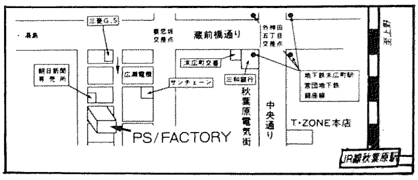
XGA採用パーソナルディスクトップ
80386SX(20MHz)2MB or 6MB
パソコンらしくないパソコン？
オシャレ

386SX搭載のスタンダードPC
DOSV.VGA、ATバス採用
80386SX(20MHz)2MB or 6MB

PS/55Z 最強パフォーマンスPC
XGA高解像度・大容量
80386(20MHz)2MB or 6MB

T·ZONE お茶の水 PS/FACTORY

TEL:03-3257-2750
FAX:03-3257-2751
Net Wareデモ中！ 何でもご相談下さい。



ディスプレイディスパッチドライバー by C·F·Computing

今話題のS3ボードやTiGAボードに付属の英語ドライバーで日本語が使える!! 特価¥3,800

VGA DOS/Vの必須アイテムは早いVGAアダプタです。

ORCHID FAHRENHEIT 1280 1MB ¥89,800　ATI GRAPHICS VANTAGE 1MB ¥89,800

ATI GRAPHICS ULTRA 1MB ¥158,000
STB WIND/X(マウス付) 1MB ¥82,800
STB 1280 1MB ¥108,000
STB 1280 2MB ¥138,000
ET-4000 1MB ¥29,800

CREATIVELABS
マルチメディア対応のSOUND BLASTERは必需品です！ 特価提供中
SOUND BLASTER

これ1つでマルチメディアが楽しめます。
内蔵CD-ROM
SOUND BLASTER PRO
MIDI KIT
マルチメディアイクステンション他ソフト付
UPGRADE KIT ~~¥104,800~~

ステレオ22FM音源
CD-ROMインターフェイス他
SOUND BLASTER互換
Multimedia Sound Standard
PRO ~~¥57,800~~

11FM音源
MIDIジョイスティックパワーアンプマイク入力 各種ソフト付
V2.0 ~~¥34,800~~

PS/55等のマイクロチャンネル版です。
MCA ~~¥64,800~~

VGAモニター

DOS/V対応やWINDOWS用として高解像大画面のモニターがほしいあなた、何を選びますか？

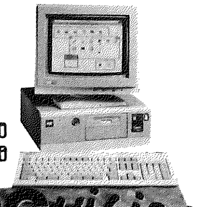
NANAO T660i（ケーブル別）
20"0.31mm 30～78kHz 1280×1024 ~~¥598,000~~

三菱 XC-98V3
14"0.28mm 98/VGA 640×480 ¥78,000

TATUNG(マイコン搭載)
17" 0.28mm 29～65kHz 1280×1024
超特価 ¥180,000

AST RESEARCH JAPAN K.K. ASTから DOS/Vマシン

新登場

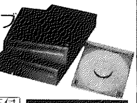


80386SX/20MHz
FDDモデル ~~¥308,000~~
40MBモデル ~~¥355,000~~

● 高速32ビットCPUを採用。メインメモリ2MBを標準搭載
● 80387数値演算プロセッサの搭載が可能。
● IBM PC ATスロットを5基装備。

PCvision

マルチメディア対応高速CD-ROMドライブ



● 標準SCSI(最大7台まで同時接続可能)
● 高速アクセス(340ms)、64KBキャッシュ内蔵
● CD-ROM XA対応
● 市販オーディオディスク演奏可能

	ドライブのみ	AT I/F付	PS/2 I/F付
内蔵モデル	¥125,000	¥145,000	¥158,000
外付モデル	¥135,000	¥155,000	¥168,000

特価提供中

DOS/Vノートおすすめセット 限定

● CPU：80386SX/20MHz
● RAM：3MB
● FDD：3.5"×1
● HDD：40MB
● 外部VGA出力
● A4ファイルサイズ

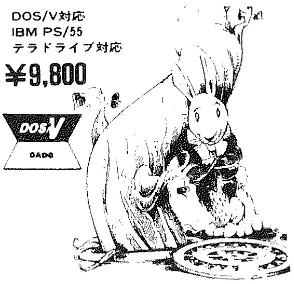
＋ DOS J5.0/V ＋ シリアルマウス ＋ 一太郎 dash

何と！何と！の 超特価¥278,000

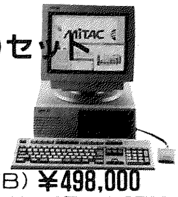
the Manhole（日本語版）

それはマンホールからはじまった……
ゲームを超えたファンタジーアドベンチャー発売中！
DOS/V対応
IBM PS/55
テラドライブ対応
¥9,800

DOS/V OADG



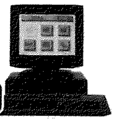
MiTAC 4060G-124-8M(JP)セット



メモリ：8MB
HDD：120MB
3.5インチFDD×1
5インチFDD×1
M1420モニター付SVGA(1MB) ¥498,000

T·ZONEがあなたに送る 超大特価テラドライブセット（モニター付）

MODEL 1セット ¥79,800
MODEL 2セット ¥89,800
MODEL 3セット ¥118,000



DOS/V版 パソコン電子手帳くん ¥14,800 新登場

シャープの電子手帳とDOS/Vマシンの間でデータのやりとりができるようになりました。
対応している電子手帳
PA6000、6500、7000、7500、8500、8600シリーズ。

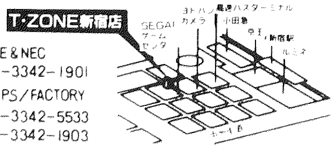
80386/33MHz セット 超特価¥398,000
RAM 4MB　64KBキャッシュ　3.5"×1、5"×1　80MB HDD　スーパーVGA
2シリアル、1パラレル　1ゲーム　マウス　101キー　1MB SVGAモニター

Microsoft BALLPOINT MOUSE 超特価¥28,000

VGA信号をビデオに変換するコンバーター 超特価¥98,000
320×200～640×480の信号をS-VHS、ビデオに変換して、テレビやビデオに接続できます。

MINI-SCSI 超特価¥29,800
プリンタポートにつなぐだけでSCSIデバイスが使えるようになります。
87 MB HDDセット 超特価¥98,000

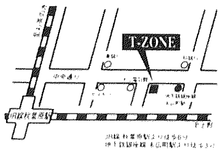
T·ZONE 新宿店 ☎03-3342-5533



1F APPLE & NEC 03-3342-1901
2F IBM PS/FACTORY 03-3342-5533
FAX 03-3342-1903
〒160 東京都新宿区西新宿1-12-18

T·ZONE 本店 ☎03-3257-2650

2F MICOMゾーン ☎03-3257-2649
4F PS PLAZA
3F IBMゾーン ☎03-3257-2652
FAX 03-3257-2661



〒101 千代田区外神田4-4-1

T·ZONE 上尾店 ☎048-774-0511

スーパー型パソコンストアー
上尾にオープン！
営業時間 AM10:30～PM7:00

〒362 埼玉県上尾市緑丘3-5-28

☆T・ZONEの7Fは専門書籍（国内・外）・データーブックの専門フロアです。 大量在庫中！ ほしい本は何でも揃います。

〈資料請求番号002〉

8086インストラクション一覧

フラグレジスタ（FLAGSの左から）
- OF：オーバーフローフラグ
- DF：ディレクションフラグ
- IF：割り込み許可フラグ
- TF：トラップフラグ
- SF：サインフラグ
- ZF：ゼロフラグ
- AF：補助キャリーフラグ
- PF：パリティフラグ
- CF：キャリーフラグ

※記号の意味
- －：影響されない
- x：変化する
- ?：不定になる
- 1：セットされる
- 0：クリアされる

オペランド
- reg：汎用レジスタ
- mem：メモリ
- acc：アキュムレータ
- imm：イミディエイトデータ
- sreg：セグメントレジスタ

※記号の意味
- ×：不可
- ○：ワードオペランド使用可能
- ◎：バイトまたはワードオペランド使用可能

データ転送命令

ニーモニック	書式	機能	FLAGS
MOV	MOV <dest>,<src>	<src>で指定されるオペランドの内容を<dest>で指定されるオペランドに転送する	---------
XCHG	XCHG <dest>,<src>	<dest>と<src>の内容を交換する	---------
PUSH	PUSH <src>	<src>で指定されるオペランドの内容をスタックに退避する	---------
POP	POP <dest>	スタックから取り出した内容を<dest>で指定されるオペランドに転送する	---------
XLAT	XLAT	DS:BXに置かれた変換テーブルに従い，ALレジスタの内容を変換する	---------
IN	IN <dest>,<port>	<port>で指定されるI/Oポートの内容を<dest>で指定されるオペランドに転送する	---------
OUT	OUT <port>,<src>	<port>で指定されるI/Oポートに<src>で指定されるオペランドの内容を転送する	---------
LEA	LEA <dest>,<src>	<src>で指定されるオペランドのオフセットを<dest>で指定されるオペランドに転送する	---------
LDS	LDS <dest>,<src>	<src>で指定されるダブルワードのポインタのうちセグメントセレクタをDSレジスタに，オフセットを<dest>で指定されるオペランドに転送する	---------
LES	LES <dest>,<src>	<src>で指定されるダブルワードのポインタのうちセグメントセレクタをESレジスタに，オフセットを<dest>で指定されるオペランドに転送する	---------
LAHF	LAHF	フラグワードの下位バイトの内容をAHレジスタに転送する	---------
SAHF	SAHF	フラグワードの下位バイトにAHレジスタの内容を転送する	----xxxxx
PUSHF	PUSHF	スタックにフラグワードの内容を退避する	---------
POPF	POPF	スタックから取り出した内容をフラグワードに転送する	xxxxxxxxx

MOV

dest＼src	reg	mem	acc	imm	sreg
reg	◎	◎	×	◎	○
mem	◎	×	◎[1]	◎	○
acc	×	◎[1]	×	×	×
imm	×	×	×	×	×
sreg	○[2]	○[2]	×	×	×

XCHG

dest＼src	reg	mem	acc	imm	sreg
reg	◎	◎	○	×	×
mem	◎	×	×	×	×
acc	○	×	×	×	×
imm	×	×	×	×	×
sreg	×	×	×	×	×

PUSH

<src> reg	mem	acc	imm	sreg
○	○	×	◎[3]	○

POP

<dest> reg	mem	acc	imm	sreg
○	○	×	×	○[4]

IN

<dest> reg	mem	acc	imm	sreg
×	×	◎	×	×

OUT

<src> reg	mem	acc	imm	sreg
×	×	◎	×	×

LEA

dest＼src	reg	mem	acc	imm	sreg
reg	×	◎[5]	×	×	×
mem	×	×	×	×	×
acc	×	×	×	×	×
imm	×	×	×	×	×
sreg	×	×	×	×	×

LDS/LES

dest＼src	reg	mem	acc	imm	sreg
reg	×	○[6]	×	×	×
mem	×	×	×	×	×
acc	×	×	×	×	×
imm	×	×	×	×	×
sreg	×	×	×	×	×

注1：ダイレクトアドレッシングのみ
注2：CSレジスタを除く
注3：バイトデータはワードデータに符号拡張される
注4：CSレジスタは除く
注5：オペランドのサイズは意味を持たない
注6：ダブルワード

算術演算命令

ニーモニック	書式	機能	FLAGS
ADD	ADD <dest>,<src>	<dest>と<src>によって指定される2つのオペランドの内容を加算し，その結果を<dest>で指定されるオペランドに転送する	x---xxxxx
ADC	ADC <dest>,<src>	<dest>と<src>によって指定される2つのオペランドの内容およびCFの内容を加算し，その結果を<dest>で指定されるオペランドに転送する	x---xxxxx
INC	INC <op>	オペランドの内容に1を加算する	x---xxxx-
AAA	AAA	1バイトの10進数どうしの加算結果を1バイトの10進数に補正する	?---??x?x
DAA	DAA	1バイトのパックBCDどうしの加算結果をパックBCD形式に補正する	?---xxxxx
SUB	SUB <dest>,<src>	<dest>で指定されるオペランドの内容から<src>で指定されるオペランドの内容を減算し，その結果を<dest>で指定されるオペランドに転送する	x---xxxxx
SBB	SBB <dest>,<src>	<dest>で指定されるオペランドの内容から<src>で指定されるオペランドの内容およびCFを減算し，その結果を<dest>で指定されるオペランドに転送する	----xxxxx
DEC	DEC <op>	<op>で指定される内容より1減算する	x---xxxx-
NEG	NEG <op>	<op>で指定される内容の2の補数をとる	x---xxxxx
CMP	CMP <dest>,<src>	<dest>と<src>によって指定される2つのオペランドの内容を比較する	x---xxxxx
AAS	AAS	1バイトの10進どうしの減算結果を1バイトの10進数に補正する	?---??x?x
DAS	DAS	1バイトのパックBCDどうしの減算結果をパックBCD形式に補正する	?---xxxxx
MUL	MUL <src>	バイトまたはワードデータの符号なし乗算を行う	x---????x
IMUL	IMUL <src>	バイトまたはワードデータの符号付き乗算を行う（1オペランド形式）	x---????x
	IMUL <dest>,<src1>,<src2>	バイトまたはワードデータの符号付き乗算を行う（3オペランド形式）	
AAM	AAM	2進数から10進数への変換を行う	x---xx?x?
DIV	DIV <src>	"ワード÷バイト"または"ダブルワード÷ワード"の符号なし除算を行う	?---?????
IDIV	IDIV <src>	"ワード÷バイト"または"ダブルワード÷ワード"の符号付き除算を行う	?---?????
AAD	AAD	あとに実行される除算命令のために，2バイト2桁の10進数を2進数に変換する	?---xx?x?
CBW	CBW	ALレジスタのバイトデータをワードに符号拡張する	---------
CWD	CWD	AXレジスタのワードデータをダブルワードに符号拡張する	---------

ADD/ADC/SUB/SBB/CMP

dest＼src	reg	mem	acc	imm	sreg
reg	◎	◎	×	◎[1]	×
mem	◎	×	×	◎[1]	×
acc	×	×	×	◎	×
imm	×	×	×	×	×
sreg	×	×	×	×	×

INC/DEC/NEG

<op> reg	mem	acc	imm	sreg
◎	◎	×	×	×

MUL/IMUL(1オペランド形式)/DIV/IDIV

<src> reg	mem	acc	imm	sreg
◎	◎	×	×	×

IMUL(3オペランド形式)

<dest> reg	mem	acc	imm	sreg
○	×	×	×	×

<src1> reg	mem	acc	imm	sreg
○	○	×	×	×

<src2> reg	mem	acc	imm	sreg
×	×	×	◎	×

注1：ワードオペランドに対してバイトのイミディエイトデータを操作することができる．この場合バイトデータは16ビットに符号拡張された上で操作される．

ストリング命令

ニーモニック	書式	機能	FLAGS
MOVS	MOVSB	メモリ間転送を行う（オペランドのバイト指定）	---------
	MOVSW	メモリ間転送を行う（オペランドのワード指定）	
	MOVS <dummy1>,<dummy2>	メモリ間転送を行う（<dummy1>，<dummy2>によるサイズ指定）	
CMPS	CMPSB	メモリ間のデータの比較を行う（オペランドのバイト指定）	x---xxxxx
	CMPSW	メモリ間のデータの比較を行う（オペランドのワード指定）	
	CMPS <dummy1>,<dummy2>	メモリ間のデータの比較を行う（<dummy1>，<dummy2>によるサイズ指定）	
SCAS	SCASB	メモリとアキュムレータの比較を行う（オペランドのバイト指定）	x---xxxxx
	SCASW	メモリとアキュムレータの比較を行う（オペランドのワード指定）	
	SCAS <dummy>	メモリとアキュムレータの比較を行う（<dummy>によるサイズ指定）	
LODS	LODSB	メモリ中のデータをアキュムレータに転送する（オペランドのバイト指定）	---------
	LODSW	メモリ中のデータをアキュムレータに転送する（オペランドのワード指定）	
	LODS <dummy>	メモリ中のデータをアキュムレータに転送する（<dummy>によるサイズ指定）	
STOS	STOSB	アキュムレータのデータをメモリに転送する（オペランドのバイト指定）	---------
	STOSW	アキュムレータのデータをメモリに転送する（オペランドのワード指定）	
	STOS <dummy>	アキュムレータのデータをメモリに転送する（<dummy>によるサイズ指定）	

リピート・プリフィックス（ストリング命令と組み合わせて使用）

ニーモニック	機能	FLAGS
REP	MOVS命令，STOS命令，INS命令，OUTS命令の前に付けることにより，これらの命令を繰り返し実行する	– – – – – – – – –
REPE/REPZ	CMPS命令，SCAS命令の前に付けることにより，これらの命令を繰り返し実行する	– – – – – – – – –
REPNE/REPNZ	CMPS命令，SCAS命令の前に付けることにより，これらの命令を繰り返し実行する	– – – – – – – – –

論理演算命令

ニーモニック	書式	機能	FLAGS
NOT	NOT 〈op〉	オペランドのすべてのビットを反転する	– – – – – – – – –
AND	AND 〈dest〉,〈src〉	2つのオペランドのビットごとの論理積をとり，演算結果を〈dest〉で指定されるオペランドに転送する	0 – – – x x ? x 0
OR	OR 〈dest〉,〈src〉	2つのオペランドのビットごとの論理和をとり，演算結果を〈dest〉で指定されるオペランドに転送する	– 0 – – x x ? x 0
XOR	XOR 〈dest〉,〈src〉	2つのオペランドのビットごとの排他的論理和をとり，演算結果を〈dest〉で指定されるオペランドに転送する	0 – – – x x ? x 0
TEST	TEST 〈dest〉,〈src〉	2つのオペランドのビットごとの論理積をとる，この演算結果は破棄されフラグのみが変化する	0 – – – x x ? x 0

AND/OR/XOR/TEST

dest＼src	reg	mem	acc	imm	sreg
reg	◎	◎	×	◎	×
mem	◎	×	×	◎	×
acc	×	×	×	◎	×
imm	×	×	×	×	×
sreg	×	×	×	×	×

NOT

〈op〉 reg	mem	acc	imm	sreg
◎	◎	×	×	×

シフト・ローテート命令（FLAGSのOFは〈count〉に1を指定した場合のみ変化する）

ニーモニック	書式	機能	FLAGS
SHL/SAL	SHL 〈op〉,〈count〉	〈op〉で指定されるオペランドを〈count〉で指定される回数だけ左シフトする	– – – – – – – – –
	SAL 〈op〉,〈count〉	〈op〉で指定されるオペランドを〈count〉で指定される回数だけ左シフトする	
SHR	SHR 〈op〉,〈count〉	〈op〉で指定されるオペランドを〈count〉で指定される回数だけ論理右シフトする	x – – – x x ? x x
SAR	SAR 〈op〉,〈count〉	〈op〉で指定されるオペランドを〈count〉で指定される回数だけ算術右シフトする	0 – – – x x ? x x
ROL/ROR	ROL 〈op〉,〈count〉	〈op〉で指定されるオペランドを〈count〉で指定される回数だけ左ローテートする	– – – – – – – – x
	ROR 〈op〉,〈count〉	〈op〉で指定されるオペランドを〈count〉で指定される回数だけ右ローテートする	
RCL/RCR	RCL 〈op〉,〈count〉	〈op〉で指定されるオペランドおよびCFを〈count〉で指定される回数だけ左ローテートする	x – – – – – – – x
	RCR 〈op〉,〈count〉	〈op〉で指定されるオペランドおよびCFを〈count〉で指定される回数だけ右ローテートする	

SHL / SAL / SHR / SAR / ROL / ROR / RCL / RCR

〈op〉 reg	mem	acc	imm	sreg
◎	◎	×	×	×

分岐命令

ニーモニック	書式	機能	FLAGS
CALL	CALL 〈op〉	〈op〉で指定されるサブルーチンに分岐する	– – – – – – – – –
RET	RET	CALL命令によって呼び出されたサブルーチンから制御を戻す	– – – – – – – – –
	RET 〈op〉	CALL命令によって呼び出されたサブルーチンから制御を戻し，SPレジスタに〈op〉が加算される	
JMP	JMP 〈op〉	〈op〉で指定されるロケーションに分岐する	– – – – – – – – –
Jcondition	JE 〈op〉	ZFがセットされている場合，〈op〉で指定されるロケーションに分岐する	– – – – – – – – –
	JZ 〈op〉	ZFがセットされている場合，〈op〉で指定されるロケーションに分岐する	
	JL 〈op〉	SFとOFが等しくない場合，〈op〉で指定されるロケーションに分岐する	
	JNGE 〈op〉	SFとOFが等しくない場合，〈op〉で指定されるロケーションに分岐する	
	JLE 〈op〉	SFとOFが等しくない，またはZFがセットされている場合，〈op〉で指定されるロケーションに分岐する	
	JNG 〈op〉	SFとOFが等しくない，またはZFがセットされている場合，〈op〉で指定されるロケーションに分岐する	
	JB 〈op〉	CFがセットされている場合，〈op〉で指定されるロケーションに分岐する	
	JNAE 〈op〉	CFがセットされている場合，〈op〉で指定されるロケーションに分岐する	
	JBE 〈op〉	CFがセットされている，またはZFがセットされて場合，〈op〉で指定されるロケーションに分岐する	
	JNA 〈op〉	CFがセットされている，またはZFがセットされて場合，〈op〉で指定されるロケーションに分岐する	
	JP 〈op〉	PFがセットされている場合，〈op〉で指定されるロケーションに分岐する	
Jcondition	JPE 〈op〉	PFがセットされている場合，〈op〉で指定されるロケーションに分岐する	– – – – – – – – –
	JO 〈op〉	OFがセットされている場合，〈op〉で指定されるロケーションに分岐する	
	JS 〈op〉	SFがセットされている場合，〈op〉で指定されるロケーションに分岐する	
	JNE 〈op〉	ZFがクリアされている場合，〈op〉で指定されるロケーションに分岐する	
	JNZ 〈op〉	ZFがクリアされている場合，〈op〉で指定されるロケーションに分岐する	
	JNL 〈op〉	SFとOFが等しい場合，〈op〉で指定されるロケーションに分岐する	
	JGE 〈op〉	SFとOFが等しい場合，〈op〉で指定されるロケーションに分岐する	
	JNLE 〈op〉	SFとOFが等しく，かつZFがクリアされている場合，〈op〉で指定されるロケーションに分岐する	
	JG 〈op〉	SFとOFが等しく，かつZFがクリアされている場合，〈op〉で指定されるロケーションに分岐する	
	JNB 〈op〉	CFがクリアされている場合，〈op〉で指定されるロケーションに分岐する	
	JAE 〈op〉	CFがクリアされている場合，〈op〉で指定されるロケーションに分岐する	
	JNBE 〈op〉	CFとZFがともにクリアされている場合，〈op〉で指定されるロケーションに分岐する	
	JA 〈op〉	CFとZFがともにクリアされている場合，〈op〉で指定されるロケーションに分岐する	
	JNP 〈op〉	PFがクリアされている場合，〈op〉で指定されるロケーションに分岐する	
	JPO 〈op〉	PFがクリアされている場合，〈op〉で指定されるロケーションに分岐する	
	JNO 〈op〉	OFがクリアされている場合，〈op〉で指定されるロケーションに分岐する	
	JNS 〈op〉	SFがクリアされている場合，〈op〉で指定されるロケーションに分岐する	
	JCXZ 〈op〉	CXレジスタが0の場合，〈op〉で指定されるロケーションに分岐する	
LOOP	LOOP 〈op〉	CXレジスタから1を引いた結果，0にならなければ〈op〉で指定されるロケーションに分岐する	– – – – – – – – –
LOOPE/ LOOPZ	LOOPE 〈op〉 LOOPZ 〈op〉	ZFがセットされていてかつCXレジスタから1を引いた結果，0にならなければ〈op〉で指定されるロケーションに分岐する	– – – – – – – – –
LOOPNE/ LOOPNZ	LOOPNE 〈op〉 LOOPNZ 〈op〉	ZFがクリアされていてかつCXレジスタから1を引いた結果，0にならなければ〈op〉で指定されるロケーションに分岐する	– – – – – – – – –

割り込み命令

ニーモニック	書式	機能	FLAGS
INT/INTO	INT 〈op〉	〈op〉で指定されるベクトルの割り込みを発生する	– 0 0 – – – – – –
	INTO	OFがセットされていたら，オーバーフロー割り込みを発生する	
IRET	IRET	割り込み処理ルーチンからもとのプログラムに制御を戻す	– 0 0 – – – – – –

プロセッサ制御命令

ニーモニック	書式	機能	FLAGS
STC	STC	CFをセットする	– – – – – – – – 1
CLC	CLC	CFをクリアする	– – – – – – – – 0
CMC	CMC	CFを反転する	– – – – – – – – x
STD	STD	DFをセットする	– 1 – – – – – – –
CLD	CLD	DFをクリアする	– 0 – – – – – – –
STI	STI	IFをセットする	– – 1 – – – – – –
CLI	CLI	IFをクリアする	– – 0 – – – – – –
HLT	HLT	CPUをホールド状態にする	– – – – – – – – –
WAIT	WAIT	BUSY端子がHighになるまで命令の実行を中止する	– – – – – – – – –
ESC	ESC	拡張プロセッサ命令	– – – – – – – – –
LOCK	LOCK	LOCKプリフィックスの付けられた命令の実行中，LOCK端子をアサートする	– – – – – – – – –
NOP	NOP	ノーオペレーション	– – – – – – – – –

セグメントオーバーライドプリフィックス

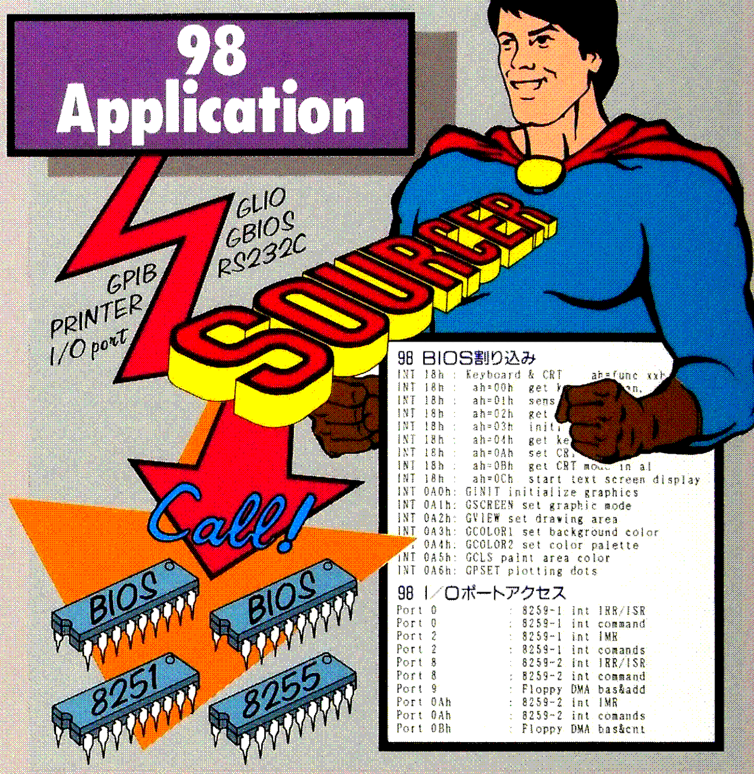
ニーモニック	書式	機能	FLAGS
segment:	CS: 〈op〉	メモリオペランドに使用されるセグメントレジスタを〈op〉によって指定されたオペランドに変更する	– – – – – – – – –
	DS: 〈op〉		
	ES: 〈op〉		
	SS: 〈op〉		

新バージョン発売中!!

LIFEBOAT

EXEファイルやCOMファイルからアセンブラソースリストを生成する究極のコード解析ツールSOURCER。さらに強化され、さらに速くなった『バージョン4』は、98のBIOSコールをも検知する新性能を搭載。

98のBIOSコールをハッキング!



Version 4.50の新しい機能

●PC-9801用SOURCERは、従来のDOSファンクションコールに加え、98のハードウェアに依存したソフトウェア割り込み（BIOSコール）、I/Oポートアクセスに関する手続きを検出。解析ソースのコメント部、あるいはクロスリファレンステーブルに発見された手続きをレポート出力します。 ＊1

●PC-9801用SOURCERの解析スピードは従来のバージョンと比べ、約5倍の高速化を実現しています。

●出力ソース形式(.ASM)は、従来のアセンブラ処理系に加え、Microsoft(MASM6.0)、Borland(TASM2.5/3.0)を新たにサポートします。

●Windows仮想デバイスドライバに組み込まれた32bitフラットなプログラムコード（従来のリアルモード用デバッガでは解析不能）を自動検知。最新の32bitアセンブラコードに変換します。

Windows仮想デバイスドライバとは?
EMM386.EXE v86mmgrh.386 vddh.386 windebug.386 などがこれに相当します。

●DOS5.0やDPMI(Dos Protected Mode Interface)、VCPI(Virtual Control Program Interface)、Windowsファンクション(INT 20h VMMファンクションなど)のコメント出力がサポートされました。

●文字列を参照しているコードのコメント部にそのテキスト内容をインライン表示することができます。

●ハードウェアテーブルのサポートにより、未知の割り込み処理やI/Oポートアクセスに関するコメントをユーザが定義することも可能です。

●98用およびPC/AT用以外のSOURCERでは、エスケープシーケンスによるスクリーンオペレーションを、5から最高10倍まで高速化しました。 ＊2

●また、PC-9801用SOURCERは、従来のエスケープ描画をVRAM直接表示とし、ユーザインターフェースを向上しました。＊3

●その他、新たなデフィニッションファイルオプション、起動オプション等が追加されています。

＊1)98用およびPC/AT用以外のSOURCERはBIOS.I/Oポートのコメント出力はサポートされていません。＊2)98用およびPC/AT用以外のSOURCERは従来のエスケープシーケンス描画によりスクリーンオペレーションを行います。＊3)PC/AT用でもVRAM直接描画を行います。

SOURCERの機能と特徴

■実行形式ファイルを解析し、詳細なコメント付きのアセンブラソースリストに変換します。

■多くの場合、出力されたアセンブラソースをそのまま再アセンブルし、実行形式ファイルを再構築することができます。

■解析作業は、プログラムを実際にシミュレートすることにより、命令部／データ部の分離を実現しています。

■SOURCERは以下のファイル形式のコードを解析することができます。

- ●COMファイル ●EXEファイル
- ●オーバーレイファイル
- ●デバイスドライバ
- ●ROMやRAMにマッピングされたコード

■8086から486までの各種命令セットに対応した逆アセンブルが可能です。

■80286以上のCPUではプロテクトモードインストラクションも解析できます。

■出力ソースは以下のアセンブラにそのままかけられます。

- ●Microsoft Macro Assembler
- ●Borland Turbo Assembler
- ●OPTASM

■出力ソースにDOSファンクションの他、様々なコメントを自動的に埋め込みます。

■デフィニッションファイルにより、ユーザが任意に解析方法を指示することができます。

■デバイスドライバやレジデントプログラムのエントリポイントを検出します。

■各種アイテムのクロスリファレンスを出力します。

■各種アセンブラのバグを回避するようなソースを出力します。

■EMSドライバ(LIM EMS4.0)の併用により、大きなプログラムを解析することができます。

お知らせ!

ただ今、各種サービスのお申し込みを受け付けております。

■バージョンアップサービス
郵送にてお知らせの通り、正式登録ユーザの方のバージョンアップのお申し込みを受け付けております。お申し込みは下記まで、お早めにご連絡下さい。
お申し込み料金¥10,000(消費税別)6/末まで有効

■グレードアップサービス
米国版SOURCER(IBM PC用)をお使いだった方は、今回に限り、特別価格で弊社の販売するSOURCER version 4.5をお求め頂けます。ご希望の方は、SOURCERをお使いになられる機種名をご指定の上、下記までお申し込み下さい。
特別価格¥28,000(消費税別) 7/末まで有効

『ザ・プログラマーズ・インデックス』 TEL:03-3293-3887
受け付け時間：土・日・祭日を除く9:00から17:00まで

Commenting Disassembler
SOURCER Ver.4.5
新登場

価格¥38,000(消費税別)

●お求めの際は、SOURCERをご使用になる機種名とメディアタイプをご指定ください。
●SOURCERのお求めは、お近くのパソコンソフト取扱店へ。

【対応機種】

対応機種	対応DOS	対応モード
PC-9801	MS-DOS 2.11/3.X/5.0	日本語
FMR J-3100 AX	MS-DOS 3.X/5.0	日本語
PS/55	DOS/V 4.0/5.0	日本語
PC/AT	PC-DOS 3.X/4.X/5.0	英語
必要メモリ	各機種640Kバイト(EMS使用可)	

株式会社LIFEBOAT
〒101 東京都千代田区神田錦町3-6
PHONE:03-3293-4711㈹ FAX:03-3293-4710

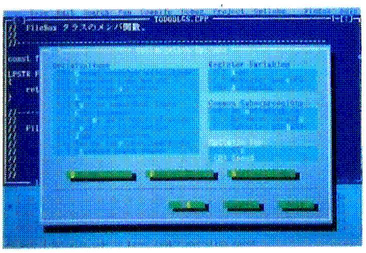
＊SOURCERは(米)V Communication, Inc.の商標です。＊その他、プログラム名、CPU名、システム名は一般に各メーカの(登録)商標です。＊コンピュータソフトウェアは、法律により著作権が保護されています。逆アセンブルの禁止されたソフトウェアに対しては、SOURCERを使用しないで下さい。

〈資料請求番号003〉

BORLAND
The Leader in Object-Oriented Programming

世界標準C&C++コンパイラ

BORLAND C++ & APPLICATION FRAMEWORKS 3.0(以下BC++ & AFX 3.0)は、プロ向けの超高速最適化コンパイラです。ANSI CとAT&T C++ 2.1をサポートし、テンプレートも使えます。強力な最適化機能によりコードを速く、小さくすることも可能になりました。豊富なツール類がDOSとWindowsの大規模なアプリケーション開発を強力に支援します。



●BC++の最適化オプション画面

■強力な最適化機能搭載の新・C&C++コンパイラ

広範囲にわたる革新的な最適化が最小のコードと最高の実行速度を提供します。

■プログラマーをサポートするさまざまな機能

BC++ & AFX 3.0には、最適化後のコードのデバッグも可能なTurbo Debugger、オブジェクト指向に拡張されたTurbo Assembler、Windowsリソースの編集ができるResource Workshop、Windows対応したプログラムのボトルネックを調べるTurbo Profiler、Windowsメッセージを監視するWinSight、DOSの標準入出力をWindows上でエミュレートするEasyWin、Windows上の新統合環境TC++ for Windows 3.0など、便利なツール類や、従来は別売だった標準ライブラリとAFXのソースコードまでもが含まれています。もちろんコンパイルスピードも飛躍的に向上しており、BC++ & AFX 3.0だけで、DOSとWindows上のあらゆるアプリケーションが快適に開発できます。

■製品内容…マニュアル13冊・ディスク13枚

■動作環境…●使用機種:NEC PC-9800シリーズ(ハイレゾモード対応、XAを除く)、EPSON 286/386シリーズ(CPU 80286以上のマシン)●メモリ:要プロテクトメモリ・1MB以上(推奨5MB以上)●OS:MS-DOS Ver.3.3以上、MS-Windows Ver.3.0以上(Windowsアプリケーション作成時のみ)●周辺機器:要ハードディスクドライブ・15MB以上の空き領域(推奨容量50MB)、バスマウス対応

■価格…………………………¥98,000(税別)

ご注意:VCPIまたはDPMIに対応していないプロテクト・メモリを使用するアプリケーション(EMSドライバなど)とは共存できません。

「新生ボーランド キックオフ・キャンペーン」プレゼント当選者発表!!

東京都/石田雅彦様　北海道/及川博志様
神戸市/田島斌光様　大阪市/濵口　徹様
川崎市/原田浩樹様

●以上5名の方に、「NEC PC-9801 NC40」をお送りします。おめでとうございます!

BORLAND® C++
& APPLICATION FRAMEWORKS 3.0

ボーランド株式会社 〒151 東京都渋谷区笹塚1-64-8 笹塚サウスビル TEL.03-5350-9370㈹ FAX.03-5350-9390

※All Borland Products are trademarks or registered trademarks of Borland International, Inc ©Borland International, Inc
※Microsoft, Windowsは米国マイクロソフト社の登録商標です。その他、商品名は一般に各社の商標です。

※詳しくは、カタログをご請求ください。

T1017777060983　〈資料請求番号P04〉　雑誌コード17777-06